Resum

La memòria presentada a continuació és un recull de l'estudi previ i els passos que s'han seguit en aquest treball de final de grau. És un recorregut per les parts més importants del projecte.

El primer bloc consistirà en un breu resum de tot l'estudi previ realitzat sobre el hardware i el software que s'ha considerat en la realització del projecte, indispensable per a un correcte disseny de la biblioteca.

En un segon bloc trobareu el resum del desenvolupament de la biblioteca usbTinLib, que ha estat dissenyada per facilitar la comunicació amb un bus CAN des d'un ordinador mitjançant un adaptador USB-CAN USBtin. La biblioteca controla el dispositiu a través de l'enviament de comandes mitjançant un port sèrie (USB) i aquest executa les accions pertinents sobre el bus. A més de la biblioteca s'ha dissenyat una interfície gràfica amb la qual facilitar encara més l'ús de la biblioteca substituint les comandes al terminal per elements visuals i gràfics.

Per últim, es troba la síntesis de la documentació realitzada per facilitar l'ús de la biblioteca així com per entendre el funcionament de l'aplicació. Aquesta documentació serà molt útil per a futures modificacions de terceres persones, ja que simplificarà la comprensió del codi.

La intenció de realitzar el projecte residia en la necessitat de personalitzar certes característiques disponibles a l'aplicació de l'enginyer Thomas Fischl – creador de l'USBtin – per tal de poder treballar amb més d'un dispositiu des de la mateixa aplicació. Per fer-ho ha calgut refer la biblioteca de mètodes sencera en un llenguatge com és Python, que té l'avantatge de ser més accessible per a tots els usuaris.

Sumari

ETSEIB

1. Glossari

1.1. Llista d'abreviatures/acrònims

DLC: Data Length Code / Número de bytes de dades

GUI: Graphical User Interface / Interfície Gràfica d'Usuari

ID: Identificador (referit a missatges CAN)

IDE: Integrated Development Environment / Entorn de Desenvolupament Integrat

PC: Personal Computer / Ordinador Personal

RTR: Remote Transmission Request / Sol·licitud de Transmissió Remota

S.O.: Sistema Operatiu

1.2. Llista de definicions/Termes anglesos

Bitrate: Taxa de bits per segon. És una mesura de la velocitat de transmissió de dades a la xarxa i indica la quantitat de bits que poden ser rebuts o enviats en un segon.

Baudrate: Taxa d'unitats de senyal per segon. És una mesura de la velocitat de transmissió de dades a la xarxa i indica la quantitat d'unitats de senyal. que poden ser rebuts o enviats en un segon.

****Unitat de senyal:*** Una unitat de senyal és la codificació que s'ha fet sobre els bits (una unitat de senyal podria ser la codificació per nivell alt o baix de tensió). És habitual que els mitjans de transmissió dels senyal digitals estiguin limitats a una freqüència determinada i fa que la codificació clàssica dels bits en nivells alts o baixos no sigui possible. D'aquesta manera s'han de trobar codificacions alternatives (unitats de senyal) com podria ser associar bits a determinats rangs de freqüències. D'aquesta manera es poden trobar codificacions on s'inclogui més d'un bit. (P.ex. associant 1 kHz a 00, 2 kHz a 01, 3 kHz a 10 i 4 kHz a 11).

(Bitrate/Baudrate: Conceptes diferents a la teoria com s'ha definit anteriorment. En canals CAN, però, la transmissió es produeix a través de la codificació en nivell alt i baix de tensió. La unitat de senyal (la codificació) només admet dos valors: o 0 o 1. Per tant, en tot el projecte, baudrate i bitrate prenen el mateix valor i poden ser considerats sinònims.)

Binding: Implementació d'una biblioteca genèrica per un llenguatge de programació

particular.

Buffer: Espai de memòria dedicat a l'emmagatzematge temporal de dades.

Firmware: Bloc d'instruccions que estableix la lògica a un nivell molt baix d'un dispositiu i té interacció directa amb el dispositiu al qual serveix.

***Frame*:** 1. *(Referit a GUI)* Contenidor dels objectes bàsics (widgets) amb els que interaccionarà l'usuari. 2. *(Referit a transmissió de dades)* Bloc de dades.

Hardware: Components o parts físiques d'un sistema electrònic.

***Script*:** Programa que conté un codi d'instruccions en un determinat llenguatge de programació.

Software: Part d'un projecte que engloba tot el codi i els programes per al correcte funcionament de la part física o hardware.

String: Tipus de variable en diferents llenguatges de programació representada per una cadena de caràcters.

Thread: Fil d'execució d'un determinat script.

Toctree: Arbre de continguts.

Widget: Objecte visual i funcional bàsic del mòdul tkinter que es pot vincular a determinades comandes o accions.

1.3. Llista de Figures

1.4. Llista de Taules

2. Introducció

CAN és un protocol de comunicacions àmpliament estès en l'actualitat però a pesar del seu èxit la majoria d'eines desenvolupades al mercat tenen uns costos poc assumibles per a empreses de poc pressupost. El desenvolupament d'aplicacions de codi lliure faciliten l'accés de l'usuari a eines que fins al moment quedaven reservades per a empreses amb un mida mitja o gran. Aquest projecte treballa utilitzant un gran nombre d'eines de codi lliure i, de fet, el propi dispositiu USBtin és un hardware lliure. El desenvolupament d'aquest projecte ha estat indirectament col·laboratiu i la intenció és que també pugui formar part de futurs projectes.

2.1. Objectius del projecte

El propòsit del projecte té diverses vessants. Per una banda tenim una visió més pràctica i aplicada en què l'objectiu clar és el de crear una biblioteca de mètodes que ens permeti enviar i rebre missatges del PC a un bus CAN a través d'un port sèrie amb un o més dispositius USBtin. També s'ha de dissenyar una aplicació GUI (interfície gràfica d'usuari) que faciliti encara més l'ús d'aquesta biblioteca sense que l'usuari hagi de navegar pel terminal. Tot això desenvolupat en un llenguatge com és Python, que permeti a un usuari amb poca experiència programant poder utilitzar-ho i modificar-ho sense dificultats. Amb la biblioteca desenvolupada es pot crear, per exemple, un datalogger de bus CAN de baix cost amb diferents canals així com generar tràfic de missatges CAN en diferents bussos per fer test i validació de xarxes ja existents.

Per altra banda hi ha una vessant més teòrica i personal on l'objectiu principal és el de desenvolupar coneixements en l'àmbit de la programació, una habilitat que a l'era actual és imprescindible en el món laboral.

2.2. Projectes previs

Hi ha tres precedents que han servit com a model i referents al llarg de tot el projecte. El primer d'ells és la biblioteca USBtinLib dissenyada en Java pel també pel creador de l'USBtin, Thomas Fischl. Aquesta biblioteca té exactament la mateixa finalitat que la que es vol crear en aquest projecte, però en un llenguatge més inaccessible per l'usuari mig i que dificulta la personalització d'aplicacions com la interfície d'usuari. No obstant això, l'existència d'aquesta biblioteca ha estat molt útil per estructurar la nova biblioteca en Python.

El segon projecte amb el qual s'ha intentat seguir certa coherència és la biblioteca

dissenyada pel meu company de l'ETSEIB, Marcel Garrido (USBCAN). Gràcies a aquest projecte s'han trobat solucions similars als problemes que plantejava el disseny de certs mètodes.

El tercer i últim projecte que s'ha considerat és la biblioteca CANLib de Kvaser, amb la qual s'ha mantingut un paral·lelisme en la nomenclatura de funcions i constants sempre que ha estat possible.

2.3. Abast del projecte

El projecte té un ventall de possibilitats molt gran, ja que ens permet monitoritzar qualsevol bus CAN i participar activament en ell mitjançant l'enviament de missatges. El protocol CAN s'utilitza especialment en la indústria automobilística però no és l'únic àmbit ni molt menys. Això sumat al fet de dissenyar la biblioteca en Python, un llenguatge prou familiar per a un gran nombre d'usuaris, àmplia les possibilitats d'ús. També el fet de treballar amb una eina barata com és l'USBtin fa més accessible tot el projecte

A més, el fet de treballar amb un dispositiu en constant desenvolupament fa que sigui possible augmentar les opcions de la biblioteca en un futur, obrint la porta a nous projectes.

2.4. Planificació temporal

Abans de començar el projecte es va definir un petit esquema temporal que s'ha seguit al llarg del treball. Tot i que el gràfic no representa el total d'hores dedicades, si que dóna una idea de quan s'han iniciat certes tasques.

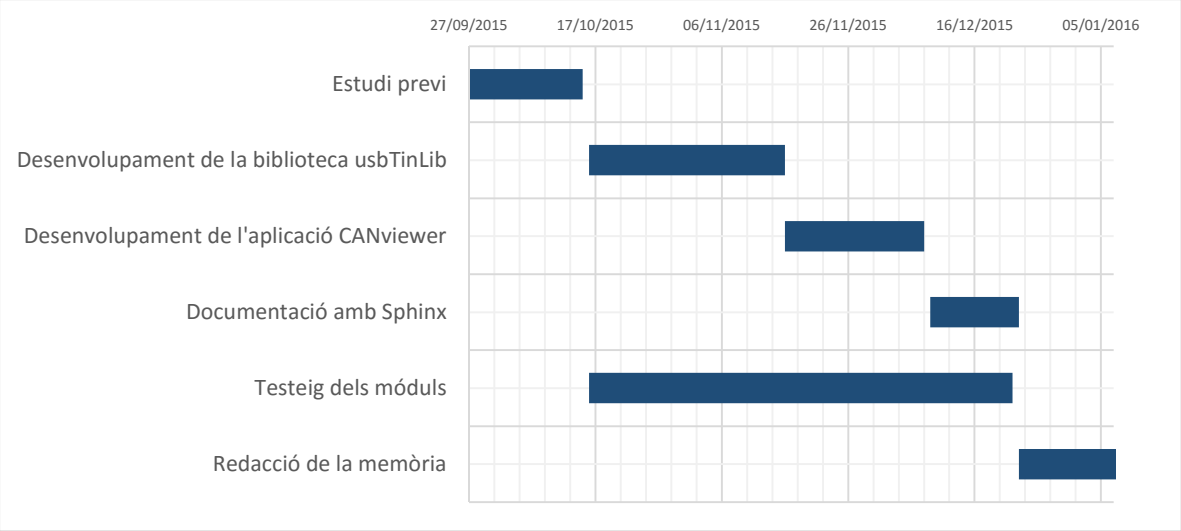


FIGURA 1 – Diagrama de Gantt de la planificació temporal del projecte. Font pròpia.

3. Hardware del projecte

3.1. USBtin

L'USBtin és una adaptador/conversor CAN-USB. Permet processar senyals rebudes d'un bus CAN i transmetre-les a un altre dispositiu (en el cas tractat un PC) a través d'un port sèrie tipus USB. També en el sentit contrari és d'utilitat, ja que ens permet generar missatges al bus donada una ordre des del dispositiu.

Trobem diferents configuracions segons la versió de USBtin que tinguem a les mans. A la figura 2 es pot observar la versió 2.0 (fotografia superior) i la versió 1.2 (fotografia inferior). La versió actual és més compacta perquè utilitza components muntats en superfície (SMD) mentre que l'antiga té components de forats passants (through hole). En les dues versions trobem tres circuits integrats: un que serveix com a interfície entre els cables físics del bus CAN i un controlador CAN (MCP2551) que ens permet a més configurar el valor del bitrate de la comunicació amb el bus CAN (fins a 1 Mbit/s), un altre circuit integrat que funciona com el controlador CAN abans especificat que implementa el protocol CAN (MCP2515). És el que permet en un nivell més baix executar les ordres per enviar i rebre missatges CAN. A més, té dos buffers que permeten l'emmagatzematge d'una quantitat limitada de dades. L'últim dels circuits integrats és un microcontrolador que s'encarrega de gestionar la comunicació sèrie (PIC18F14K50) i transmetre les comandes que rep als altres circuits. És sobre aquest microcontrolador on es carrega el firmware (codi d'instruccions a baix nivell que defineix la lògica interna). També trobem una sèrie d'elements passius (resistències, condensadors, etc.) que complementen algunes entrades dels circuits per al seu correcte funcionament.

Quant a la comunicació amb elements externs, l'USBtin consta de dues entrades: un connector USB (femella mini-A en la versió més moderna i femella tipus-B en l'antiga) i tres pins que connecten amb el bus CAN - un del senyal CAN High, un del senyal CAN Low i un per la massa (Ground). Mitjançant els tres pins es pot aconseguir que l'USBtin sigui un node del bus CAN mentre que connectant-lo amb un dispositiu desitjat a través d'un cable USB mascle es simularà un port sèrie un cop connectat. Degut a la necessitat d'aquesta simulació per la correcte comunicació PC-USBtin, no es podrà aprofitar totes les capacitats de la connexió USB, limitant la velocitat de transmissió de dades.

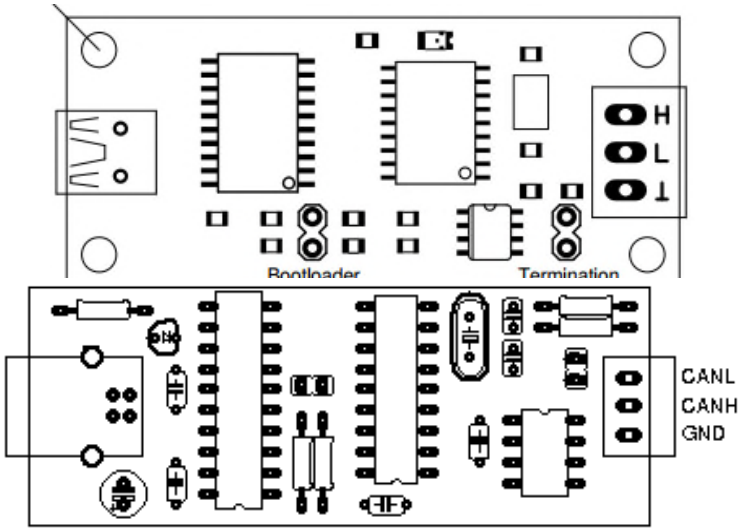


FIGURA 2 - Diferents configuracions/versions de la PCB. Font: USBtin ***[1]***

3.1.1. Comunicació PC/USBtin/CAN bus

La comunicació a través del port sèrie es realitza enviant una sèrie de comandes a l'aparell que les processa executant les accions pertinents. Aquestes comandes no són més que caràcters ASCII (lletres i números) codificats en bytes i acabats sempre en el caràcter del retorn de carro "\r", que és el que identifica el final d'una comanda. Totes aquestes accions sempre rebran alguna resposta de l'USBtin, ja sigui un byte d'error perquè l'acció no s'ha pogut dur a terme o un byte codificat amb el retorn de carro.

Donat que no podem controlar la comunicació amb el bus CAN directament, les comandes serveixen per enviar missatges al bus CAN. L'USBtin interpreta la comanda escollida, construeix el missatge CAN seguint tot el protocol i envia bit a bit la successió d'impulsos elèctrics que defineixen el missatge. El projecte de la biblioteca a construir és precisament automatitzar i facilitar l'enviament d'aquestes comandes reduint la despesa de temps.

Per últim també és molt important parlar de les velocitats o baudrates. Al bus CAN es pot escollir un ampli ventall de possibilitats segons convingui però al final la velocitat a la qual podrem processar els missatges vindrà limitada per la velocitat del port sèrie, que és de 115200 bits/s i està fixada en el firmware del microcontrolador del USBtin.

3.2. Altres

A més de l'ànima del projecte com és l'USBtin, s'han utilitzat altres elements a l'hora de testejar les biblioteques. Per simular un node del bus CAN s'ha utilitzat també el adaptador/conversor CAN/USB Kvaser Leaf Light(i el seu software).

4. Software del projecte

4.1. Llenguatge de Programació

Un dels objectius del projecte era modificar la biblioteca creada per Thomas Fischl de manera que satisfés les nostres necessitats. Inicialment, es van considerar dues opcions.

La primera d'elles va ser la d'aprofitar la biblioteca en Java ja creada, estenent-la i modificant-la amb Jython, una implementació de Python dissenyada en Java que permet treballar amb mòduls d'aquest llenguatge de manera similar al que passa amb la implementació oficial i el llenguatge C. Aquesta opció va ser descartada per diversos motius; entre ells el fet de que hi hagués certs problemes de compatibilitat i la desconeixença del llenguatge Java que haguessin requerit una inversió de temps massa alta.

La segona opció i amb la que finalment es va decidir tirar endavant va ser la de reconstruir des de zero la biblioteca USBtin i a partir d'ella construir una GUI personalitzada. El fet de comptar amb biblioteques de referència com la del ja nomenat Thomas Fischl i la d'en Marcel Garrido van ajudar a accelerar el procés, tot i haver-hi diferències en el llenguatge i l'estructura de la biblioteca. Aquesta reconstrucció es va decidir realitzar-la en el llenguatge de programació Python.

4.1.1. Python vs Java

Python és un llenguatge multiparadigma interpretat enfront a Java, que és un llenguatge orientat a objectes i compilat. Els paradigmes de programació són els recursos i les eines que té un llenguatge per enfrontar-se als problemes des de perspectives i enfocs diferents convertint a Python en un llenguatge més potent en aquesta vessant. Per altra banda, el fet que sigui un llenguatge interpretat fa que Python presenti uns temps d'execució lleugerament més alts que els llenguatges compilats com Java o C++. Això és degut a que un programa interpretat ha de traduir el codi línia a línia a llenguatge màquina (un llenguatge que pugui entendre el processador) cada vegada que és executat mentre que un llenguatge compilat fa aquesta traducció abans, generant un arxiu executable que ja conté les instruccions en un llenguatge màquina o que requereix d'una traducció més ràpida.

No obstant això, el fet que Python compti amb un intèrpret ràpid i eficient fa que aquestes diferències es minimitzin. A més a més, la biblioteca plantejada de manera efectiva reduirà en gran mesura l'exigència sobre el processador reduint al màxim les diferències de rendiment amb els llenguatges compilats.

Tot això sumat a que Python és un llenguatge on prima la claredat del codi i presenta una

estructura molt més amigable amb el desenvolupador el fa un llenguatge molt accessible i la decisió idònia. Qualsevol futura modificació del projecte es podrà veure facilitada per l'alta accessibilitat d'aquest llenguatge. No és casualitat, doncs, que grans gegants de la indústria tecnològica com Google s'hagin interessat per aquest llenguatge.

Per finalitzar és important dir que s'ha treballat amb la versió 3.5.0 de Python a l'hora de desenvolupar el programa. Això sumat al fet que el mòdul de Tkinter que es veurà més endavant fa que l'aplicació sigui compatible únicament amb versions de Python3.

4.2. Entorn de Programació - Notepad++ i l'IDLE Python

Un cop escollit el llenguatge calia escollir les eines amb les que desenvolupar les biblioteques necessàries. Cal destacar que totes les eines utilitzades són gratuïtes.

Com a editor de textos es va escollir Notepad++ (enfront a altres editors com Emacs) per la seva senzillesa i facilitat de configuració. Compta amb suport específic per múltiples llenguatges (entre ells Python) i proveeix d'una estructura de codi individualitzada a cada llenguatge un cop s'ha definit. A més permet tenir diversos arxius oberts al mateix temps en la mateixa finestra i en pestanyes diferents, cosa que facilita molt la feina en cas de tenir diversos arxius i haver d'anar canviant entre ells.

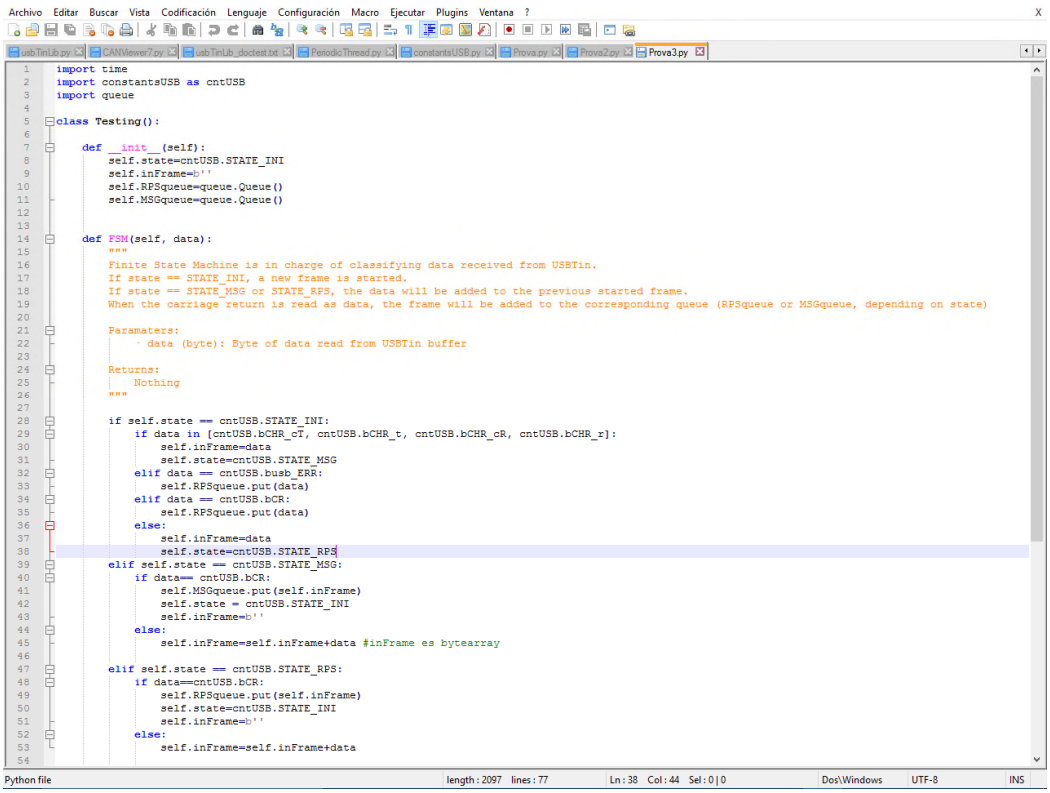


FIGURA 3 - Interfície del programa Notepad++. Font pròpia.

Per altra banda, l'IDLE de Python és un entorn de desenvolupament integrat (IDE) que fa a la vegada d'editor de textos i de testejador dels scripts. En aquest cas, s'ha fet servir l'IDLE per provar i testejar les línies de codi necessàries (tot i que a vegades s'ha testejat directament des del terminal). Això ha permès provar dos programes a la vegada ja que es poden obrir diverses finestres sobre els que obrir múltiples programes.

4.3. Biblioteques externes

Per a la realització del projecte s'ha necessitat treballar amb diverses biblioteques externes (algunes pròpies de Python), que han ajudat en la construcció de les biblioteques pròpies[1].

4.3.1. USBtinLib (biblioteca de Thomas Fischl)

USBtinLib (no confondre amb la biblioteca sobre la que tracta aquest projecte, usbTinLib) és la biblioteca dissenyada en Java per facilitar l'ús de l'USBtin. Un dels objectius del projecte, com s'ha dit, és crear una biblioteca similar en Python i poder-la implementar en una GUI personalitzada. No obstant això, s'ha pres com a model aquesta biblioteca encara que sense fer-la servir directament (principalment per la incompatibilitat dels llenguatges).

4.3.2. PySerial

PySerial ha estat la biblioteca de referència per construir la biblioteca pròpia del projecte, usbTinLib. PySerial sintetitza i facilita l'accés i la comunicació a través d'un port sèrie en qualsevol plataforma (donat que suporta MAC OS, Windows i sistemes Unix) **[2]**.

Aquesta biblioteca és, doncs, amb la que podrem controlar l'USBtin i els seus paràmetres. Quan es vol accedir a un determinat port es crea un objecte del tipus classe sobre el qual es poden cridar múltiples mètodes. Els més interessants són els de enviar (write) i rebre missatges (read), ja que són les que permetran enviar les successives comandes que també ajudin a comunicar-se a través del bus CAN.

4.3.3. Tkinter

Tkinter és una implementació (*binding)* per Python de la biblioteca Tk GUI Toolkit per al

[1] ***Nota!:*** Dins els arxius del projecte NO s'inclouen aquestes biblioteques. El terminal (ordinador) des d'on es cridi la biblioteca principal de l'arxiu (usbTinLib) o la GUI (CANUSBViewer) ha de tenir descarregades aquestes biblioteques externes. Són fàcilment instal·lables com a extensions de Python.

llenguatge Tcl. Actualment és la eina per generar GUI estàndard de Python. Amb les versions més actuals ja ve aquesta extensió instal·lada.

La construcció d'interfícies en Tkinter es basa en l'ús de frames i widgets. Aquests últims són els objectes bàsics amb els que l'usuari podrà interaccionar (botons, llistes expandibles, entrades de text, etc.) mentre que les frames són objectes "transparents" per a l'usuari i són els que contenen diferents widgets.

Tot això sota el paraigües d'un gestor de geometria que interaccionant amb els frames i les widgets permeten organitzar al gust del desenvolupador la interfície desitjada. Tkinter posa a disposició de l'usuari tres gestors de geometria: pack, grid i place. Els dos primers faciliten molt la feina a l'usuari ja que estalvien fer qualsevol tipus de càlcul a l'usuari ja que el tercer implica una definició manual en termes de posició absoluta dels objectes. Per la seva banda el mètode pack s'utilitza per definir la posició relativa d'objectes (ja siguin widgets o contenidors) a l'interior d'un contenidor mestre. Per últim el mètode grid també defineix la posició relativa d'objectes dins un contenidor però dividint-lo virtualment en columnes i files de manera que es puguin col·locar els objectes en les posicions desitjades.

L'esperit modulable de Tkinter permet aprofitar objectes en diferents parts de la GUI o fins i tot en GUIs diferents. Hi ha moltes maneres de fer-ho. Dissenyar una frame amb els objectes desitjats i vincular-ho a un objecte de Python del tipus classe és una bona manera de tenir sempre accessible un determinat disseny.

Com es veurà més endavant aquest ha estat el mètode utilitzat a la realització de la interfície gràfica en aquest projecte. Dissenyar petits mòduls genèrics i finalment incloure'ls tots sota una classe principal que serà la que es cridarà quan s'iniciï el programa.

4.3.4. Altres biblioteques

Altres biblioteques incloses a Python també han estat utilitzades com la biblioteca Queue o Time. La primera ha permès crear les cues per alliberar càrrega sobre el buffer de l'USBtin processant i guardant les dades dels missatges rebuts. La segona s'ha utilitzat per dues coses: mesurar el temps que trigaven certes accions en executar-se i postergar la realització de certes tasques.

Una altra de les biblioteques utilitzades és Sphinx, que es descriurà detalladament al punt 6.

5. Mòduls del projecte

El projecte es composa de dos mòduls principals: un d'ells conté la biblioteca usbTInLib que és la que inclou tots els mètodes necessaris per controlar la comunicació sèrie amb l'USBtin. L'altre mòdul és l'aplicació GUI CANviewer que permet interaccionar amb tots els mètodes de la biblioteca d'una manera més senzilla i visual. També s'inclouen dos mòduls complementaris, PeriodicThread i constantsUSB, que a diferència d'altres mòduls/biblioteques també utilitzats com PySerial, no compten amb documentació pròpia.

5.1. PeriodicThread

Aquest mòdul inclou la classe homònima que permet iniciar threads periòdics. No és més que una classe creada a partir del mòdul threading (més concretament la classe Timer) de Python creada i compartida **[4]** per l'usuari de Github, Kris Dorosz.

Abans de continuar cal definir que un thread és un objecte (en forma de instància d'una classe en el nostre cas), que executa una determinada rutina (línies de codi) ja sigui en funcions, classes, etc.

Aquest mòdul i la classe PeriodicThread s'ha dissenyat incloent un temporitzador (mitjançant la classe Timer del propi mòdul threading). La instància s'inicialitza passant diversos paràmetres:

- Funció objectiu (*callback)*: És la funció o mètode que volem que el thread executi periòdicament.
- Període de repetició (*period*): El temps, en segons, que es desitja entre cada repetició de l'execució dins el thread de la funció objectiu.
- Nom del Thread (name) [opcional]: Afegeix un atribut a la instància amb un nom especificat per l'usuari. No té cap més funció que identificar la instància per un dels seus atributs a més del propi nom que se li ha posat a la instància.
- Arguments de la funció objectiu (**args/**kwargs*) [opcional]: Paràmetres de la funció objectiu en cas que els necessiti. En el cas d'aquest projecte, s'ha dissenyat les funcions adjuntades a la instància de manera que no es necessiti utilitzar aquest arguments.

5.2. ConstantsUSB

El mòdul "constantsUSB.py" conté una sèrie de constants utilitzades en els mòduls de la biblioteca usbTinLib i de la GUI CANUSBviewer. Aquestes constants van des de les lletres

que s'han d'incloure en l'enviament de comandes a l'USBtin fins a les constants d'error que retornarem a cada error del bus CAN passant per les constants de velocitat del propi bus.

S'ha intentat que els noms de les constants (així com el seu valor definit) siguin els mateixos que a la biblioteca de Kvaser CANLib **[13]** o a la pròpia biblioteca que en Marcel Garrido va dissenyar en un TFG previ **[11]** per mantenir una mica la coherència. Per aquesta mateixa raó s'han definit les constants en un mòdul a part i no a la pròpia biblioteca on s'han de fer servir. Així, aquestes constants poden ser accessibles en qualsevol moment i estan acotades en un espai propi.

5.3. Biblioteca usbTinLib / Class USBTin

La biblioteca usbTinLib conté la classe USBTin i tots els seus mètodes necessaris per a controlar la comunicació en sèrie amb un o més dispositius USBtin al mateix temps. A més importa diversos mòduls (constantsUSB, PeriodicThread, serial, queue i time) que s'utilitzen a la biblioteca.

Cada instància creada d'aquesta classe quedarà vinculada a un i només un dispositiu USBtin, de manera que es podran controlar diversos dispositius al mateix temps creant diverses instàncies. És important tenir en compte que un port només es pot obrir una vegada i és per aquesta raó que només hi ha una instància per dispositiu així com un dispositiu només podrà estar vinculat a una instància.

La creació d'una classe que es pugui instanciar per a cada port sèrie dóna solució a un dels objectius que s'havien proposat a l'inici del treball.

A continuació es mostra un petit esquema dels mètodes disponibles a la biblioteca. S'indica l'apartat on es troba la seva descripció al document. També es diu quina és la seva interacció amb l'usuari: transparent (l'usuari no crida en cap moment el mètode, es troba inclòs dins d'altres mètodes), semi-transparent (l'usuari pot cridar el mètode tot i que es troba inclòs dins altres mètodes) i activa (és l'usuari el que crida directament els mètodes corresponents).

Mètode	Apartat	Interacció de l'usuari	Mètodes inclosos
__init__	5.3.1	Activa (iniciant la instància corresponent)	openSerial
openSerial	5.3.2.1	Transparent	-
closeSerial	5.3.2.2	Activa	canClose

askHWversion	5.3.2.3	Activa	ReadResponse
askFWversion	5.3.2.4	Activa	ReadResponse
askUSBstatus	5.3.2.5	Activa	ReadResponse
SerialSetTimestamping	5.3.2.6	Activa	ReadResponse
ReadResponse	5.3.2.7	Semi-transparent: Es pot cridar fora d'una funció però no tindrà cap utilitat (retornarà -2)	-
canOpenChannel	5.3.3.1	Activa	canSetBitrate ReadResponse
canClose	5.3.3.2	Semi-transparent	ReadResponse
canSetBitrate	5.3.3.3	Semi-transparent	ReadResponse
canWrite	5.3.3.4	Activa	ReadResponse
canReadMessage	5.3.3.5	Activa	ReadResponse
FSM	5.3.4.1	Transparent	-
processBytes	5.3.4.2	Transparent	-

TAULA 1 – Esquema dels mètodes de la biblioteca usbTinLib

5.3.1. Mètode inicialitzador de classe (__init__)

El mètode inicialitzador de la classe conté atributs importants que han d'estar disponibles per a tots els mètodes així com variables que s'han de mantenir declarades, tot i que algun mètode hagi finalitzat.

S'ha de declarar l'atribut SerialPort, que és el que portarà el gruix de les comunicacions USBtin-PC. El que es fa és crear una instància de la classe Serial continguda a la biblioteca abans mencionada de PySerial, de manera que ens puguem comunicar amb l'USBtin mitjançant els mètodes d'aquesta classe, havent definit el port on està connectat. Altres atributs que s'han de declarar són els referents al mode i velocitat del canal de bus CAN. Aquests atributs es declaren amb els paràmetres que se li han fet arribar en iniciar la instància USBTin. També es defineixen les cues que s'utilitzaran per afegir els

missatges/respostes rebuts i les estats inicials de la màquina d'estats FSM. Per últim es crea el thread FSM, que serà el que s'encarregarà de processar contínuament els bytes que arribin al buffer de l'USBtin.

Un cop s'han declarat aquestes variables o atributs, l'últim que fa el mètode inicialitzador és cridar el mètode openSerial() per obrir el port definit amb els atributs que se li ha donat.

Els paràmetres que s'han de definir quan es crea la instància són:

- Port (*port*): Nom del port on està connectat l'USBtin amb el qual es desitja comunicar. Aquest nom varia segons quin sigui el sistema operatiu del PC.
- Velocitat del canal CAN (*CANbaud*): Se li ha de passar una de les velocitats estanditzades a l'USBtin (10, 20, 50, 100, 125, 250, 500, 800 kBaud i 1 MBaud). Aquesta velocitat es defineix mitjançant un string en el següent format: "BAUD_125K" o "canBITRATE_1M".
- Mode del canal (*mode*): S'ha de definir en quin mode actuarà l'USBtin respecte el canal CAN. Si serà un node actiu / estàndard, un node d'escolta o un node de bucle. Respectant l'ordre anterior, els strings respectius que s'haurien de definir com a paràmetre són "Normal", "Listen-Only", "Loopback".

```
def __init__(self, port, CANbaud, mode):
        #Attributes SerialPort (baudrate - only supported)
        self.SerialPort=serial.Serial()
        self.SerialPort.port=port
        self.SerialPort.baudrate=115200
        self.SerialPort.timeout=0.01
        self.SerialPort.writeTimeout=0.01

        #CAN Attributes
        self.baudrate=CANbaud
        self.mode=mode
        self.CANOC="Closed"

        #Response and Messages Queues
        self.MSGqueue=queue.Queue()
        self.RPSqueue=queue.Queue()

        #FSM attributes
        self.inFrame=b''
        self.state=cntUSB.STATE_INI

        #Threads
        self.FSMThread=thread.PeriodicThread(self.processBytes, 0.0001)

        self.openSerial()
```

5.3.2. Mètodes referits al hardware

5.3.2.1. openSerial

Estableix la comunicació entre el PC i el hardware USBtin. A més a més, es realitzen una sèrie d'accions (assegurar tancament del canal CAN, llegir i netejar els bytes del buffer de l'USBtin) per assegurar el correcte funcionament. El creador del hardware i de la biblioteca original en Java, Thomas Fischl, realitza una sèrie de passos similars **[5]** per evitar els errors que suposadament es podrien derivar de la inicialització dels circuits integrats a la placa al connectar-la al port sèrie.

També s'inicia l'objecte FSMThread, ja definit anteriorment al mètode inicialitzador.

```
def openSerial(self):
        try:
            self.SerialPort.open()
            self.SerialPort.write(bytes('\rC\r','ascii')) #Recommended
            time.sleep(0.05)                              #Recommended
            self.SerialPort.write(bytes('C\r','ascii'))      #Recommended
            self.SerialPort.read(100);             #Recommended
            self.FSMThread.start()
            time.sleep(0.1)
            print("Port is opened.")
            self.RPSqueue.queue.clear()
        except serial.SerialException:
            print("Device not found - The COM port cannot be opened.")
```

5.3.2.2. closeSerial

Tanca el canal CAN en cas que estigui obert, finalitza l'objecte FSMThread i finalment tanca la comunicació a través del port sèrie entre el PC i el hardware USBtin.

```
def closeSerial(self):
        self.FSMThread.cancel()
        self.FSMThread.join()
        if self.CANOC=="Open": self.canClose()
        self.SerialPort.close()
        print("Port is closed.")
```

5.3.2.3. askHWversion

Retorna la versió del hardware USBtin mitjançant el mètode canReadResponse() (veure apartat 5.3.2.7). Només es retorna amb èxit la versió quan el canal CAN està tancat, en cas contrari es retorna un paràmetre d'error. Això es fa per assegurar que no hi ha creuaments de respostes (explicat en l'apartat del mètode ReadResponse).

```
def askHWversion(self):
        if self.CANOC=="Closed":
            self.RPSqueue.queue.clear()
            self.SerialPort.write(bytes(cntUSB.usb_VERSION_HW+cntUSB.CR+c
            ntUSB.LF,'ascii'))
            time.sleep(0.2)
            return self.canReadResponse()
        else:
            return cntUSB.canERR_KO
```

5.3.2.4. askFWversion

Retorna la versió del firmware USBtin mitjançant el mètode ReadResponse() (veure apartat 5.3.2.7). Retorna el valor de la mateixa manera que a la funció askHWversion.

```
        def askFWversion(self):
            if self.CANOC=="Closed":
                  self.RPSqueue.queue.clear()
                  self.SerialPort.write(bytes(cntUSB.usb_VERSION_FW+cntUS
                  B.CR+cntUSB.LF,'ascii'))
                  time.sleep(0.2)
                  return self.canReadResponse()
            else:
                  return cntUSB.canERR_KO
```

5.3.2.5. askUSBstatus

Retorna la representació binària del byte d'estat del controlador de bus CAN MCP2515 (un dels que forma part de la placa USBtin). Els següents bits donen informació de l'estat:

Bit 2 – Avís d'error (Bit EWARN de MCP2515).
Bit 3 – Sobrecàrrega de dades (Bit RX1OVR or RX0OVR de MCP2515).
Bit 5 - Error-Passiu (Bit TXEP or RXEP de MCP2515).
Bit 7 – Error del Bus (Bit TXBO de MCP2515).

5.3.2.6. SerialSetTimestamping

Activa o desactiva (segons el paràmetre que se li passi) l'empremta de temps de l'USBtin. Aquesta indica, amb una paraula de 16 bits, el instant de temps en que un missatge de CAN ha arribat..

Per defecte, el timestamping està desactivat a l'iniciar una instància del USBTin. Com el procés de marcar el temps consumeix recursos (memòria) del controlador és recomanable deixar-lo desactivat en entorns de comunicació amb un gran volum de dades.

5.3.2.7. ReadResponse

El mètode ReadResponse és l'encarregat de classificar les respostes (missatges que ha enviat l'USBtin per estímul de l'usuari a través d'algun missatge previ). La diferència principal entre un missatge i una resposta al projecte és que el missatge és un conjunt de bytes que prové d'altres nodes del bus CAN mentre que la resposta és un conjunt de bytes que prové del propi USBtin amb el qual s'està treballant en un moment determinat.

Aquesta primera selecció entre missatge i resposta ja s'ha fet en el mètode recorrent FSM, com s'explica en el punt 5.3.4.1.

La funció ReadResponse, doncs, s'encarrega de llegir el primer element de la cua RPS i descodificar-lo perquè l'usuari pugui llegir-lo com a *string*. Segons el tipus de resposta, la funció retornarà una cosa o un altre:

- *Versions de firmware i hardware*: retornarà directament els bytes enviats per l'USBtin descodificats on es mostra les versions del dispositiu.
- *Estatus de l'USB*: retornarà una representació en *string* del byte d'estatus en format binari.
- *Missatge rebut i retorn de carro*: retornarà la constant canERR_OK per assenyalar que tot és correcte
- *Error de CAN*: retornarà la constant canERR_KO per assenyalar que hi ha algun problema.

Per últim però no menys important, cal destacar que aquest mètode és totalment transparent per a l'usuari. És a dir, aquest mètode mai serà cridat i avaluat directament per l'usuari perquè ja s'inclou dins d'altres mètodes que l'usuari sí que crida (askHWversion, askFWversion, askUSBstatus, ...).

```
def ReadResponse(self):

        if self.RPSqueue.empty():
            return cntUSB.canERR_NORPS
        else:
            evalRPS=self.RPSqueue.get()
            if evalRPS.decode('ascii') == 'z' or evalRPS.decode('ascii')
            == 'Z':
                print("Message received")
                return cntUSB.canERR_OK
            elif evalRPS.decode('ascii') == cntUSB.CR:
                return cntUSB.canERR_OK
            elif bytes(cntUSB.usb_READSTATUS, 'ascii') in evalRPS:
                byte_stat=evalRPS.decode()[1:3]
                bin_stat=str(bin(int(byte_stat,16)))[2:].zfill(8)
                return bin_stat
```

```
            elif bytes(cntUSB.usb_VERSION_HW, 'ascii') in evalRPS:
                return evalRPS.decode('ascii')
            elif bytes(cntUSB.usb_VERSION_FW, 'ascii') in evalRPS:
                return evalRPS.decode('ascii')
            elif evalRPS == cntUSB.usb_ERR:
                return cntUSB.canERR_KO
            else:
                return evalRPS.decode('ascii')
```

5.3.3. Mètodes referits al canal CAN

5.3.3.1. canOpenChannel

Estableix el bitrate (definit a la funció inicialitzadora) del canal CAN cridant el mètode setCanBitrate (punt 5.3.3.3) i obre la comunicació entre el canal CAN i l'USBtin, convertint aquest últim en un node del bus. Ho obre en el mode especificat quan s'inicia la instància.

```
def canOpenChannel(self):
        try:
            self.canSetBitrate(self.baudrate)
        except ValueError:
            print('Wrong Bitrate Value. Please set a proper value calling
            USBTin.canSetBitrate() method or change attribute value
            USBTin.baudrate')
            return cntUSB.canERR_PARAM

      if self.mode == "Normal":
            self.SerialPort.write(bytes(cntUSB.can_OPEN_NORMALMODE+cntUSB
            .CR+cntUSB.LF,'ascii'))
            print("CAN Channel opened in normal mode")
            self.CANOC="Open"
        elif self.mode == "Listen-only":

            self.SerialPort.write(bytes(cntUSB.can_OPEN_LISTENONLY+cntUSB
            .CR+cntUSB.LF,'ascii'))
            print("CAN Channel opened in listen-only mode")
            self.CANOC="Open"
        elif self.mode == "Loopback":

            self.SerialPort.write(bytes(cntUSB.can_OPEN_LOOPBACK+cntUSB.C
            R+cntUSB.LF,'ascii'))
            print("CAN Channel opened in loopback mode")
            self.CANOC="Open"
        else:
            print("CAN channel cannot be opened. Only -Normal-, -
            Loopback- or -Listen-only- are taken as inputs")
        time.sleep(0.1)
        print(self.ReadResponse())
```

5.3.3.2. canClose

Tanca la comunicació entre el canal CAN i l'USBtin.

```
def canClose(self):

      self.SerialPort.write(bytes(cntUSB.can_CLOSE+cntUSB.CR+cntUSB.LF,'a
      scii'))
      print("CAN Channel closed")
      time.sleep(0.1)
      self.CANOC="Closed"
      return self.ReadResponse()
```

5.3.3.3. canSetBitrate

Estableix la velocitat del canal CAN per l'USBtin. Cal tenir en compte que si hi ha diversos nodes al bus CAN, la velocitat en bauds hauria de ser la mateixa per cada node. En cas contrari es podria produir un error al bus que s'expandiria per tots els nodes.

```
def canSetBitrate(self, baudrate):
        if baudrate in cntUSB.BAUD:

            self.SerialPort.write(bytes(cntUSB.can_SETBITRATE+str(cntUSB.
            BAUD[baudrate])+cntUSB.CR,'ascii'))
            self.baudrate=baudrate

        elif baudrate in cntUSB.canBITRATE:

            self.SerialPort.write(bytes(cntUSB.can_SETBITRATE+str(cntUSB.
            canBITRATE[baudrate])+cntUSB.CR,'ascii'))
            self.baudrate=baudrate

        else:
            raise ValueError
```

5.3.3.4. canWrite

Permet enviar un missatge a través del canal CAN. Per fer-ho cal definir una sèrie de paràmetres amb els quals es cridarà la funció i es transmetrà el missatge:

- *Identificador de missatge* (msgId): Ha de ser un *string* en format hexadecimal. Cal tenir present que per a missatges d'identificador estàndard, l'identificador s'ha de

trobar comprés entre 0x0 i 0x7FF[2] mentre que en identificadors estesos s'ha de trobar entre 0x0 i 0x1FFFFFFF.

- *Número de bytes de data* (dlc): Ha de ser un enter entre 0-8 que indiqui la subseqüent xifra de bytes de data.
- *Bytes de dades* (messageList): Llista de strings on a cada posició de la llista se li assigna un *string* en format hexadecimal[2] del byte corresponent (0x0-0xFF)
- *Atributs/Tipus de missatge* (*mask): tupla d'enters que defineixen quin tipus de missatge s'està enviant:
    - *0x0002=2:* Missatge d'identificador estàndard
    - *0x0004=4:* Missatge d'identificador estès
    - *0x0001=1:* Missatge de RTR
    - *(Si no es defineix explícitament que el missatge és RTR, serà un missatge de dades normal)*

```
def canWrite(self,msgId, dlc, messageList, *mask):
       if cntUSB.canMSG_RTR in mask:
            if cntUSB.canMSG_STD in mask:
                msgId=msgId.zfill(3)
                frame_str="r"+msgId.upper()
                if messageList!=[]: return 'Remote Request Message cannot
have Data Bytes'
            elif cntUSB.canMSG_EXT in mask:
                msgId=msgId.zfill(8)
                frame_str="R"+msgId.upper()
                if messageList!=[]: return 'Remote Request Message cannot
have Data Bytes'
            elif cntUSB.canMSG_EXT in mask and cntUSB.canMSG_STD in mask:
                return "STD and EXT ID messages cannot be defined at the
same time. Please choose only one"
            else:
                return "STD or EXT ID message is not defined. Please
define it in mask"
        else:
            if cntUSB.canMSG_STD in mask:
                msgId=msgId.zfill(3)
                frame_str="t"+msgId.upper()
            elif cntUSB.canMSG_EXT in mask:
                msgId=msgId.zfill(8)
                frame_str="T"+msgId.upper()
            elif cntUSB.canMSG_EXT in mask and cntUSB.canMSG_STD in mask:
                return "STD and EXT ID messages cannot be defined at the
same time. Please choose only one"
            else:
                return "STD or EXT ID message is not defined. Please
define it in mask"
        frame_str=frame_str+str(dlc)
```

[2] A l'hora de passar els strings en format hexadecimal no s'han d'incloure els definidors '0x'

```
        for data_byte in messageList:
            frame_str=frame_str+data_byte.zfill(2)

self.SerialPort.write(bytes(frame_str+cntUSB.CR+cntUSB.LF,'ascii'))
        time.sleep(0.1)
        self.ReadResponse()
```

5.3.3.5. canReadMessage

El mètode canReadMessage és l'encarregat de donar format als missatges que han arribat des del bus CAN. Com ja s'ha dit prèviament, la primera selecció és dur a terme pel mètode FSM.

El mètode es limita a llegir i extreure de la cua MSG el primer element i retornar-lo amb el següent format de llista: [Mode de missatge, Id missatge, DLC, Bytes de dades].

```
def canReadMessage(self):
        frameList=[]
        if self.MSGqueue.empty():
            frameList.append(cntUSB.canERR_NOMSG)
            return frameList
        else:
            evalMSG=self.MSGqueue.get().decode('ascii')
            if cntUSB.CHR_cT in evalMSG:
                frameList.append("Extended")
                frameList.append(evalMSG[1:9])
                frameList.append(evalMSG[9])
                frameList.append(evalMSG[10:])
                return frameList
            elif cntUSB.CHR_t in evalMSG:
                frameList.append("Standard")
                frameList.append(evalMSG[1:4])
                frameList.append(evalMSG[4])
                frameList.append(evalMSG[5:])
                return frameList
            elif cntUSB.CHR_cR in evalMSG:
                frameList.append("Extended RTR")
                frameList.append(evalMSG[1:9])
                frameList.append(evalMSG[9])
                frameList.append(evalMSG[10:])
                return frameList
            elif cntUSB.CHR_r in evalMSG:
                frameList.append("Standard RTR")
                frameList.append(evalMSG[1:4])
                frameList.append(evalMSG[4])
                frameList.append(evalMSG[5:])
                return frameList
            else:
                frameList.append(cntUSB.canERR_PARAM)
                return frameList
```

5.3.4. Mètodes referits al processament de bytes

5.3.4.1. FSM

El mètode FSM és el que suporta el processament de diferents bytes que li arriben com a dades, ja siguin missatges o respostes. El mètode FSM simula una màquina d'estats i sempre rep com a paràmetre un byte d'informació que serà el que processarà. Aquest byte el rep mitjançant la funció processBytes (punt següent - 5.3.4.2) directament del buffer de l'USBtin. Cal prendre com a consideració prèvia que en aquest buffer intern els missatges arriben en parts (byte a byte) però ordenats. Per tant, la tasca a realitzar és la de llegir i agrupar aquests bytes segons pertanyin o no a un mateix missatge/resposta.

Per fer-ho, considerem 3 possibles estats diferenciats: l'estat inicial (STATE_INI) és en el que es troba la maquina d'estats quan cap missatge s'està reconstruint. És l'estat que s'assoleix una vegada s'ha acabat de reconstruir el missatge i el primer estat que pren quan s'inicia la biblioteca. Els altres dos estats són els relatius a la reconstrucció de un missatge (STATE_MSG) i a una resposta (STATE_RPS). L'estat que pren en un determinat instant la màquina es guarda en un atribut global de la biblioteca que pot ser cridat i modificat sempre.

Per entendre la transició d'estats a continuació hi ha un breu resum acompanyat del diagrama de la figura 4:

- STATE_INI→STATE_MSG: Si el byte de dades coincideix amb la codificació 'ascii' del primer caràcter de transmissió de missatges ('t', 'T', 'r' o 'R')
- STATE_INI→STATE_INI: Si el byte de dades coincideix amb el retorn de carro o amb el byte d'error de l'USBtin/CAN.
- STATE_INI→STATE_RPS: En cas que els supòsits anteriors (STATE_INI→STATE_MSG i STATE_INI→STATE_INI) no és compleixin.

- STATE_MSG/STATE_RPS→STATE_INI: Si el byte de dades és el retorn de carro.
- STATE_MSG/STATE_RPS→STATE_MSG/STATE_RPS: Si el byte de dades no és el retorn de carro

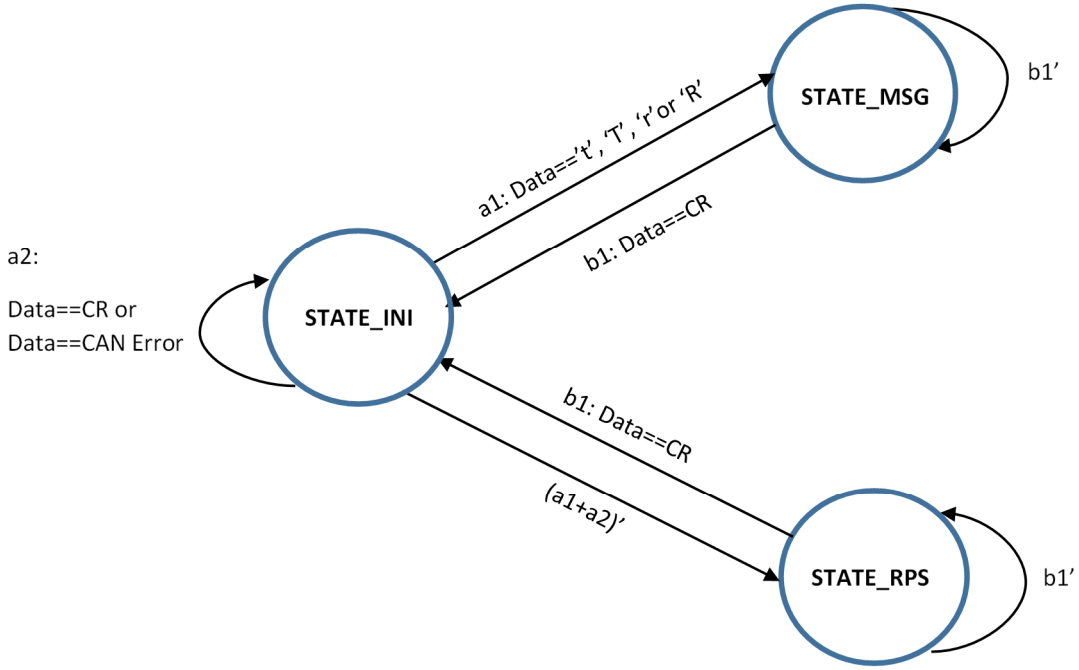


FIGURA 4 - Diagrama d'estats de la FSM. Font pròpia.

```
def FSM(self, data):
        if self.state == cntUSB.STATE_INI:
            data_str=data.decode('ascii')
            if data_str==cntUSB.CHR_cT or data_str==cntUSB.CHR_t or
            data_str==cntUSB.CHR_cR or data_str==cntUSB.CHR_r:
                self.inFrame=data
                self.state=cntUSB.STATE_MSG
            elif data == bytes(cntUSB.usb_ERR,'ascii'):
                self.RPSqueue.put(data)
                print("CAN Error")
                self.ReadResponse()
            elif data == bytes(cntUSB.CR,'ascii'):
                self.RPSqueue.put(data)
            else:
                self.inFrame=data
                self.state=cntUSB.STATE_RPS
        elif self.state == cntUSB.STATE_MSG:
            if data.decode('ascii') == cntUSB.CR:
                self.MSGqueue.put(self.inFrame)
                self.state = cntUSB.STATE_INI
                self.inFrame=b''
            else:
                self.inFrame=self.inFrame+data #inFrame es bytearray

        elif self.state == cntUSB.STATE_RPS:
            if data.decode('ascii')==cntUSB.CR:
                self.RPSqueue.put(self.inFrame)
                self.state=cntUSB.STATE_INI
                self.inFrame=b''
```

```
            else:
                self.inFrame=self.inFrame+data
```

5.3.4.1.1 Optimització FSM

La màquina d'estats és un dels mètodes crítics de la biblioteca USBtin i cal vigilar que no faci de coll d'ampolla. S'ha intentat optimitzar-lo el màxim possible per a que el temps que s'utilitza en processar la informació sigui el menor possible i la reconstrucció del missatge es faci de manera ràpida.

5.3.4.2. processBytes

L'últim mètode, processBytes, és l'encarregat de transmetre els bytes a la màquina d'estats perquè aquesta els classifiqui. És el mètode sobre el que actua el thread FSM definit en la funció inicialitzadora de la instància.

La crida a aquest mètode fa que es llegeixin un per un els bytes del buffer de l'USBtin sempre que aquest no sigui buit i es crida el mètode FSM per cada byte llegit. No s'atura el bucle fins que el buffer de l'USBtin no és buit.

El thread FSM cridarà cada 0,001 segons el mètode processBytes per veure si ha arribat algun nou byte de dades. Un cop iniciat el mètode processBytes, la crida periòdica del thread es pausarà mentre el mètode es trobà realitzant les accions dins el bucle de lectura dels bytes.

```
def processBytes(self):
    while self.SerialPort.inWaiting() != 0:
            self.FSM(self.SerialPort.read())
```

5.4. CANUSBviewer GUI

L'altre mòdul principal del projecte és el que conté la interfície gràfica (CANUSBviewer). Com ja s'ha explicat en altres apartats, l'aplicació s'ha dissenyat amb l'extensió Tkinter per Python de la biblioteca Tcl/Tk i la seva finalitat és la de treballar amb la biblioteca usbTinLib de la manera més visual i intuïtiva possible.

S'ha intentat procedir amb un disseny modular (mitjançant les classes de Python) per dues raons. La primera és la de clarificar i facilitar la comprensió del codi per a futures modificacions. La segona és la d'utilitzar la potència de les classes de Python i les avantatges que això pot comportar en el disseny de l'aplicació. Per exemple, l'avantatge de poder definir mètodes només en aquelles classes que ens interessi sense que hagin de ser

considerades variables globals. Això és ideal per adjuntar comandes a determinats botons. També el fet que es puguin repetir declaracions de la mateixa variable en diferents classes (donat que són considerades variables locals en diferents classes) beneficia en aquest cas la claredat del codi.

Cal dir que durant la construcció de cada classe s'ha considerat que tot el seu contingut anés relacionat amb un contenidor mestre ja que això ha facilitat les coses alhora d'empacar tots els petits mòduls en forma de classes. El gestor geomètric utilitzat ha estat el mètode pack (veure descripció al punt 4.3.3), cosa que ha influït a l'hora de treballar amb molts contenidors per organitzar una geometria més complexa.

Per entendre prèviament l'organització que s'ha dut a terme, a continuació es presenta un breu esquema de les classes i les subclasses (classes associades a una classe, en el projecte únicament hi ha subclasses associades a la classe USBTinFrame):

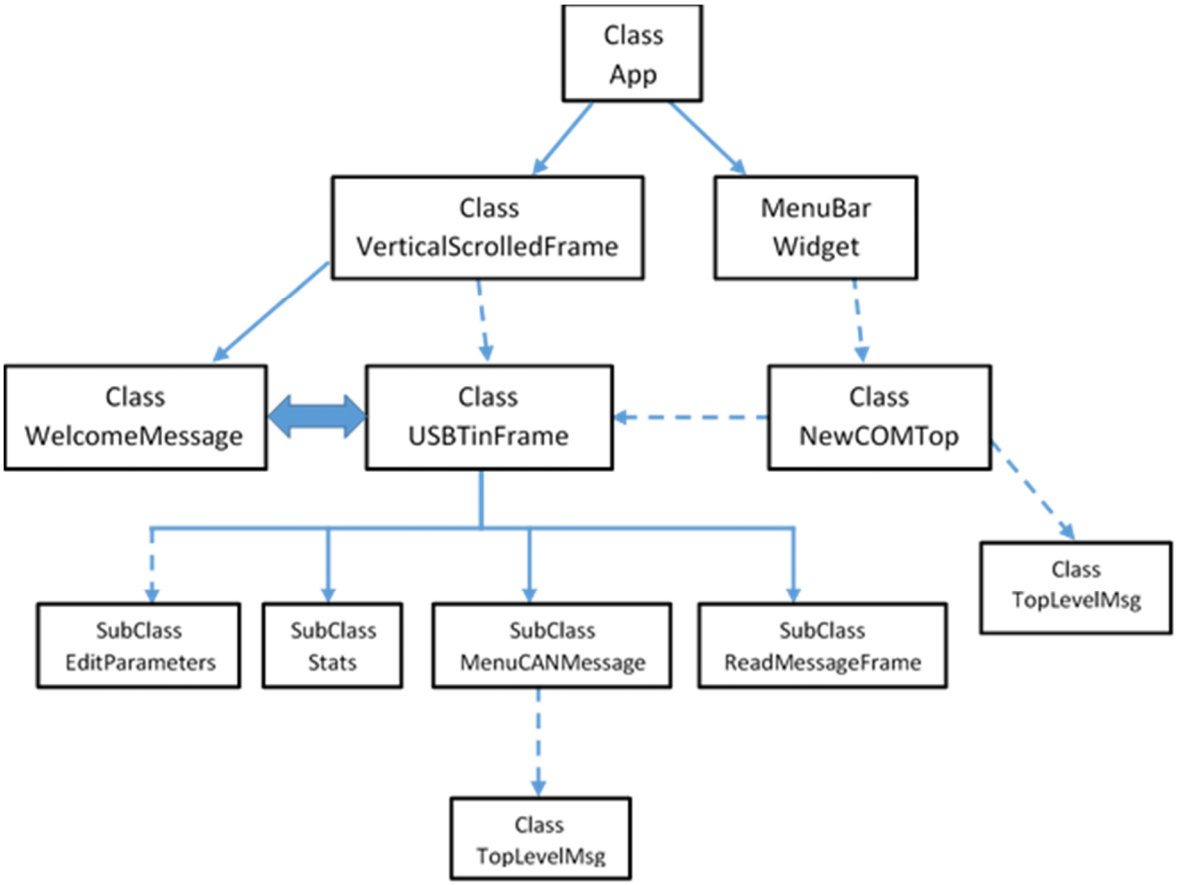


FIGURA 5 - Esquema de l'organització del codi CANUSBviewer. Font pròpia.

Tot i que les interaccions entre les classes (fletxes en l'esquema) es desenvoluparan en les descripcions que es faran a continuació de cada una d'elles, en línies generals es pot

considerar que:

- Línies continues: Impliquen la creació automàtica del contenidor mestre de la classe assenyalada per la fletxa quan la instància superior ha estat creada (Ex. Quan s'inicialitza la classe App, la classe VerticalScrolledFrame també ho fa donat que forma part del codi de la classe App).
- Línies discontinues: Impliquen la creació del frame mestre de la classe assenyalada quan l'usuari duu a terme alguna acció mitjançant alguna comanda adjunta a algun widget.
- Doble fletxa: Indica que la coexistència dels frames mestres de les dues classes alhora no és possible (Ex. Si no hi ha ports oberts serà visible el contingut del contenidor de la classe WelcomeMessage i en cas contrari el de la classe USBTinFrame)

5.4.1. Funcions: serial_ports i isUSBTin

Aquestes funcions s'han decidit mantenir fora de qualsevol classe i fer-les d'accés "global" si s'importa el mòdul per facilitar l'accés des del terminal (cosa que ha servit per incrementar la rapidesa del test).

La funció serial_ports és una modificació de la funció homònima **[6]** de Thomas Feldmann que originalment llistava tots els ports disponibles amb els quals comunicar-se via el mòdul PySerial. Aquesta modificació consisteix en avaluar si a aquests ports hi ha connectat un dispositiu USBtin mitjançant la funció isUSBTin creada especialment per aquesta finalitat. Aquesta última rep com a paràmetre el nom del port a avaluar (ex. 'COM4' al sistema operatiu Windows) i retorna el booleà *True* en cas de tenir connectat un USBtin; retorna el booleà *False* en cas contrari. Per la seva banda, la funció serial_ports retorna una llista d'aquells ports disponibles per ser oberts (és a dir, si ja estigués obert no seria considerat) i que tenen un USBtin connectat. Aquesta llista és la que llegirà la caixa d'opcions de la classe NewCOMTop (punt 5.4.5).

```
def serial_ports():
        if sys.platform.startswith('win'):
            ports = ['COM%s' % (i + 1) for i in range(256)]
        elif sys.platform.startswith('linux') or
sys.platform.startswith('cygwin'):
            # this excludes your current terminal "/dev/tty"
            ports = glob.glob('/dev/tty[A-Za-z]*')
        elif sys.platform.startswith('darwin'):
            ports = glob.glob('/dev/tty.*')
        else:
            raise EnvironmentError('Unsupported platform')

        result = []
```

```
        for port in ports:
            try:
                s = serial.Serial(port)
                s.close()
                result.append(port)
            except (OSError, serial.SerialException):
                pass
        result2=[]
        for port in result:
            if isUSBTin(port): result2.append(port)
        return result2

def isUSBTin(port):
    serial_port=serial.Serial(port=port, baudrate=115200, timeout=0.1,
writeTimeout=0.1)
    serial_port.write(bytes('\rC\r','ascii'))
      serial_port.write(bytes('C\r','ascii'))
      serial_port.read(100)
      serial_port.write(bytes(cntUSB.usb_VERSION_HW+cntUSB.CR+cntUSB.LF,'
      ascii'))
      HW=serial_port.read(10).decode('ascii')
      if HW=='': return False
      serial_port.write(bytes(cntUSB.usb_VERSION_FW+cntUSB.CR+cntUSB.LF,'
      ascii'))
      FW=serial_port.read(10).decode('ascii')
      if HW[0]=='V' and FW[0]=='v':
           serial_port.close()
           return True
      else:
           serial_port.close()
           return False
```

5.4.2. Classe App

La classe App és la que s'encarrega de generar i iniciar l'aplicació creant la finestra principal mitjançant la comanda Tk(). Aquesta finestra és l'atribut *root* de la classe App (*self.root*) ja que és un estàndard en el disseny de programes en Tkinter. I és sobre aquesta finestra sobre la que s'executa el mètode *mainloop,* que és el que inicia pròpiament l'aplicació i dóna pas a la revisió constant de tots els esdeveniments que es poden produir.

Continguts (altres classes):

- *Widget Canvas i contenidor intern – Class VerticalScrolledFrame (punt 5.4.3****¡Error! No se encuentra el origen de la referencia.****)*
- *Barra de menú:* Aquest widget està inserit directament a la finestra principal. Conté un botó ("New") que si es prem obre el contingut de la classe NewCOMTop (punt 5.4.5). No s'ha considerat fer una classe específica donat que la quantitat d'elements del menú és relativament petita (es pot definir en 3 o 4 línies de codi) i també ho és la quantitat de comandes associades als apartats d'aquest menú.

5.4.3. Classe VerticalScrolledFrame

Aquesta classe és una variant del mòdul homònim que està a disposició dels usuaris al web Unpythonic **[7].** S'han modificat els prefixos referents a mòduls importats (p.ex. a la biblioteca del GUI del projecte el mòdul Tkinter s'importava com tk) així com alguns paràmetres de les dimensions del contenidor perquè coincidissin amb les mesures desitjades dels continguts successius.

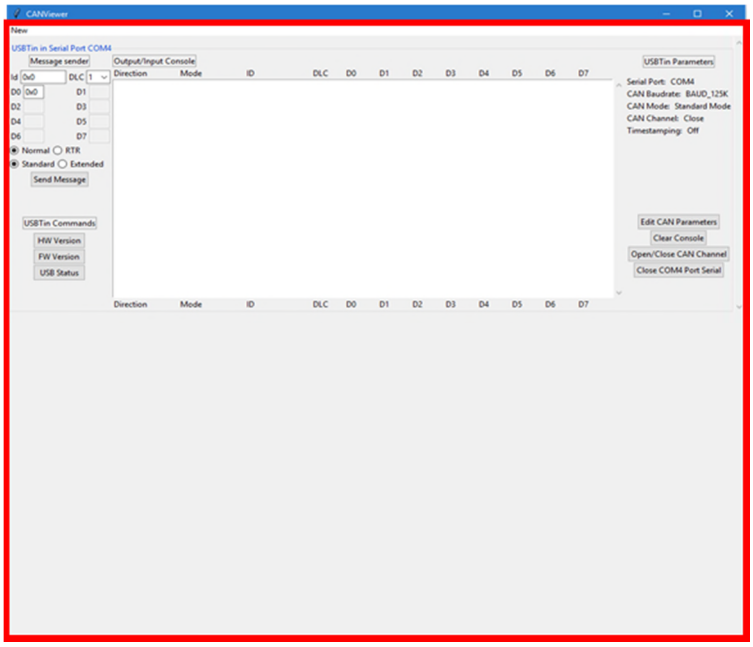
Donat que un contenidor normal (frame) no es pot desplaçar i és fixe, s'havia de trobar una solució que donés la possibilitat de navegar en vertical per la finestra. Això era necessari perquè la finestra només donava espai per a observar dos nodes al mateix temps. Si s'obria un tercer node mitjançant la classe USBTinFrame (punt 5.4.6) quedava "fora de la pantalla" amb la conseqüència de no poder interactuar amb el contenidor.

La solució que s'ha trobat amb aquesta classe és la utilització d'un widget Canvas per fer d'intermediari entre el contenidor i la barra de desplaçament. Aquest widget és utilitzat majoritàriament per mostrar gràfics i imatges i suporta l'ús de barres de desplaçament. En el nostre cas, en comptes de mostrar una imatge mostra un contenidor que gràcies a la potència del widget el podem desplaçar.

La instància de la classe VerticalScrolledFrame es crea automàticament com un dels atributs (self.Canvas) de la classe App en inicialitzar-se aquesta última. El contenidor adjunt al Canvas també es defineix com un atribut de la classe App (self.MainFrame) per a futurs usos un cop creada la instància.

Continguts (altres classes):

- *Contenidor Principal*: Aquest contenidor és el que donarà utilitat a l'aplicació. Si no hi ha cap port obert, es mostrarà un contingut de presentació i benvinguda a l'usuari indicant els passos per obrir un port (punt **¡Error! No se encuentra el origen de la referencia.**). En cas que hi hagués algun port obert es mostraria el contingut d'interacció amb aquest (punt 5.4.6).



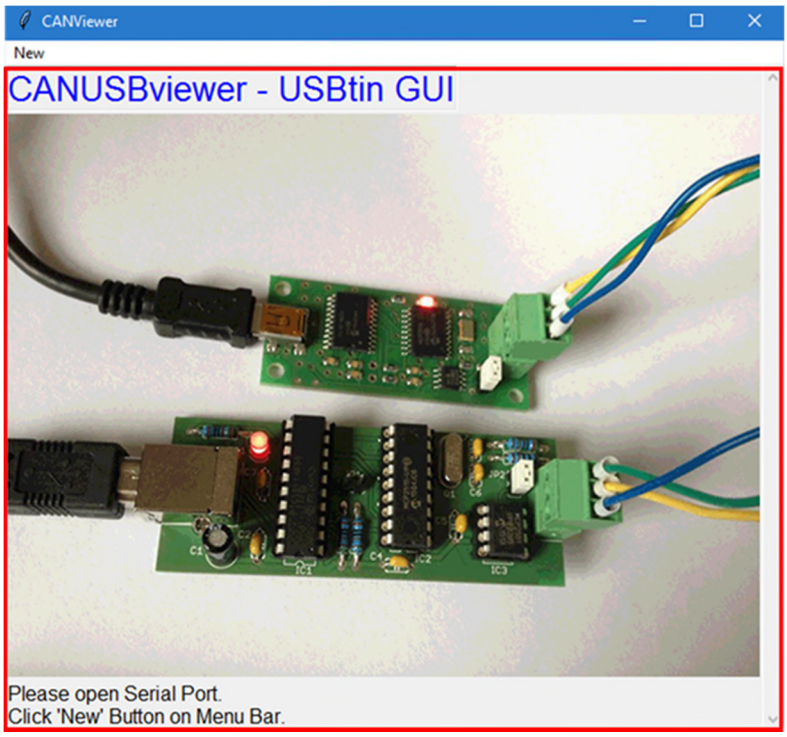


FIGURA 6 - Contenidor Classe VerticalScrolledFrame. Font pròpia.

5.4.4. Classe WelcomeMessage

La classe WelcomeMessage te la única finalitat de servir de carta de presentació i no deixar estèticament deserta la finestra en iniciar l'aplicació. El contenidor de la classe inclou el títol de l'aplicació, una imatge del dispositiu i dues línies indicant els passos per accedir a la

finestra des d'on obrir els ports.

El contenidor principal de la classe WelcomeMessage s'ha empacat dins el contenidor creat a la classe VerticalScrolledFrame (que és també l'atribut self.MainFrame). Es podria haver empacat directament tot el contingut de la classe WelcomeMessage al contenidor principal MainFrame, però per raons pràctiques s'ha desestimat aquesta opció. Això es degut a que quan s'obri algun port s'hauran de destruir tots els continguts i es més fàcil destruir un sol contenidor que uns quants petits objectes.

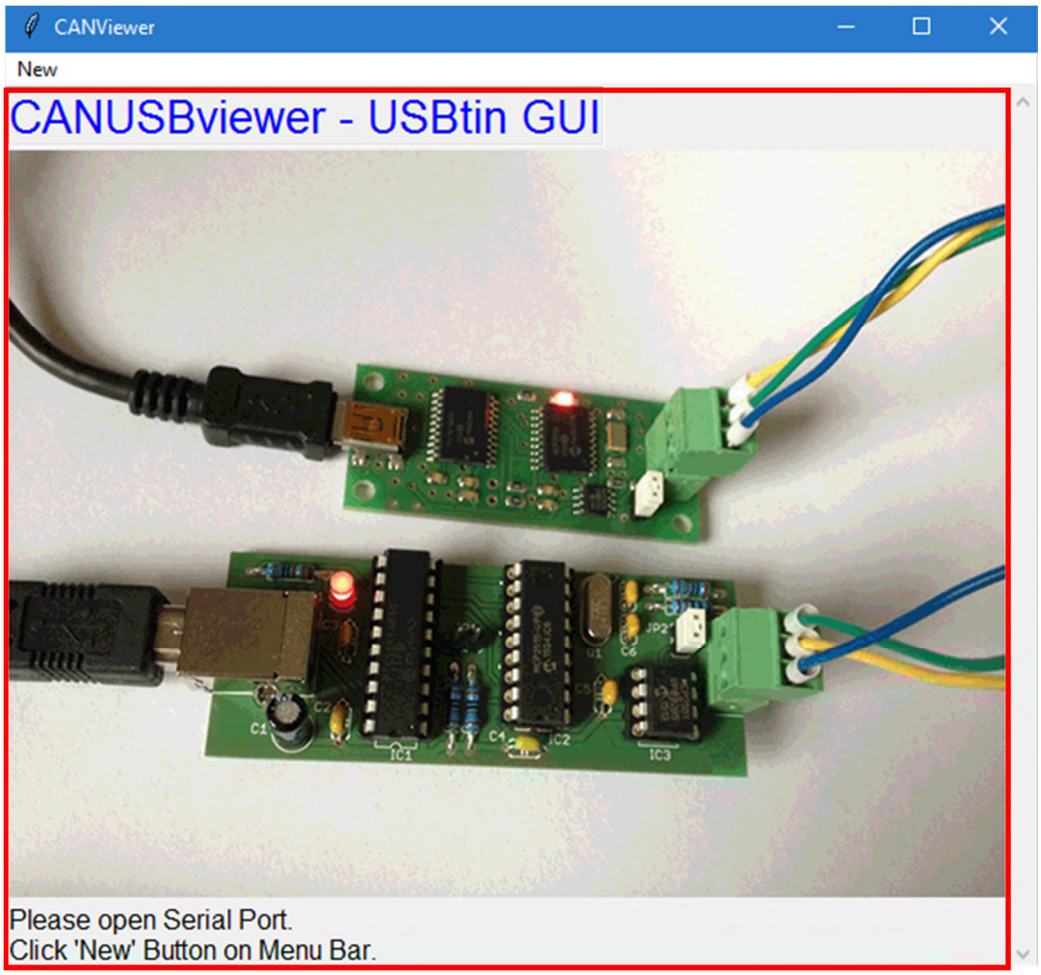


FIGURA 7 - Contenidor Classe WelcomeMessage. Font pròpia.

5.4.5. Classe NewCOMTop

La inicialització d'aquesta classe genera una nova finestra que té com a finestra pare la principal de l'aplicació (atribut self.root de la classe App).

NewCOMTop s'inicialitza quan l'usuari clica el botó "New" de la barra de menú. La finestra

creada conté tres caixes desplegables per escollir les opcions amb les que s'inicialitza la instància USBTin de la biblioteca usbTinLib. Aquestes opcions són:

- *Port*: Apareix la llista de ports disponibles per obrir generat per la funció serial_ports.
- *Velocitat del canal CAN / Baudrate*: Es pot escollir entre les5.3.1 velocitats estandarditzades per l'USBtin mencionades al punt 5.3.1.
- *Mode del node*: Paral·lelament al que s'ha explicat al punt 5.3.1, si es selecciona l'opció "Standard Mode" s'obrirà el node en mode actiu, si es selecciona "Listen-Only Mode" ho farà en mode d'escolta i si es selecciona "Loopback Mode" s'obrirà en mode de bucle.

El botó "Accept" que es troba a la dreta de totes les caixes desplegables inicialitza el contenidor USBTinFrame sempre i quan hi hagi un port vàlid per ésser obert, esborra el contenidor principal de la classe WelcomeMessage i hi empaca el contenidor de la classe USBTinFrame.

FIGURA 8 - Finestra Classe NewCOMTop. Font pròpia.

5.4.6. Classe USBTinFrame

Com ja s'ha indicat en apartats anteriors, la classe USBTinFrame s'inicialitza quan s'escullen uns determinats paràmetres (port, velocitat de canal CAN i mode del node). Amb aquests paràmetres s'inicialitza al mateix temps una instància de la classe USBTin de la nostra biblioteca usbTinLib que queda lligada com l'atribut USBLib (i d'igual manera tots els seus mètodes).

Un cop inicialitzada, el contenidor principal de la classe és lligat com a contingut de l'atribut mainFrame de la classe App, eliminant el missatge de benvinguda en cas que no hi hagués cap port obert. En cas que sí hi hagués algun port obert, la nova instància creada s'afegiria a l'antiga, de manera que veuríem els contenidors que fan referència a diferents ports apilats i podríem interactuar amb ells al mateix temps.

El contenidor principal de la instància USBTinFrame és un contenidor amb etiqueta (LabelFrame) on es numera l'USBtin utilitzat segons el port obert. Dins d'aquest contenidor s'han situat tres contenidors (veure Figura 9): el primer contenidor és el que es correspon amb el número 4 de la figura (subclasse MenuCANMessage), el segon contenidor es correspon amb el número 3 (subclasse ReadMessageFrame) i el tercer contenidor es trobaria a la dreta dels dos abans mencionats. Aquest últim té com a continguts dos contenidors: el que es correspon amb el número 2 de la figura (subclasse Stats) i un que apila els 4 botons que es veuen a la cantonada d'avall a l'esquerra.

Aquest botons s'han definit en un contenidor directament a la inicialització de USBTinFrame sense haver-lo dissenyat prèviament en un mòdul a part (subclasse) i serveixen per interactuar amb les instàncies de totes les subclasses un cop s'han cridat també a la funció inicialitzadora. El primer d'ells ("Edit CAN Parameters") permet obrir la subclasse EditParameters en una nova finestra, el segon ("Clear Console") neteja la pantalla de la consola de text de ReadMessageFrame, el tercer ("Open/Close CAN Channel") obre o tanca la comunicació amb el port CAN podent-se visualitzar al contenidor de la subclasse Stats i l'últim ("Close COM4 Port Serial") tanca la comunicació entre port serial i PC i destrueix la instància (i per tant el contenidor) de USBTinFrame.

Per últim es important mencionar que a la classe USBTinFrame s'invoca la classe PeriodicThread creant una instància que s'encarrega de visualitzar la cua de missatges de l'atribut USBLib i en cas que no estigués buida, mostrar aquest missatge per la pantalla de ReadMessageFrame (punt 5.4.6.3)

FIGURA 9 - Contenidor Classe USBTinFrame. Font pròpia.

5.4.6.1. Subclasse EditParameters

Un cop es crea la instància (prement el botó “Edit CAN Parameters”), es crea una nova finestra on apareixen diferents apartats per modificar determinats paràmetres del bus CAN com la velocitat, el mode del node respecte el canal o l’empremta de temps (que prèviament no s’havia pogut definir en inicialitzar la instància USBTinFrame). Per aplicar aquests canvis que es veuran reflexats a les etiquetes del contenidor de la subclasse Stats (punt 5.4.6.2) només cal prémer el botó “Accept”, que a més destruirà la finestra que prèviament s’havia generat.

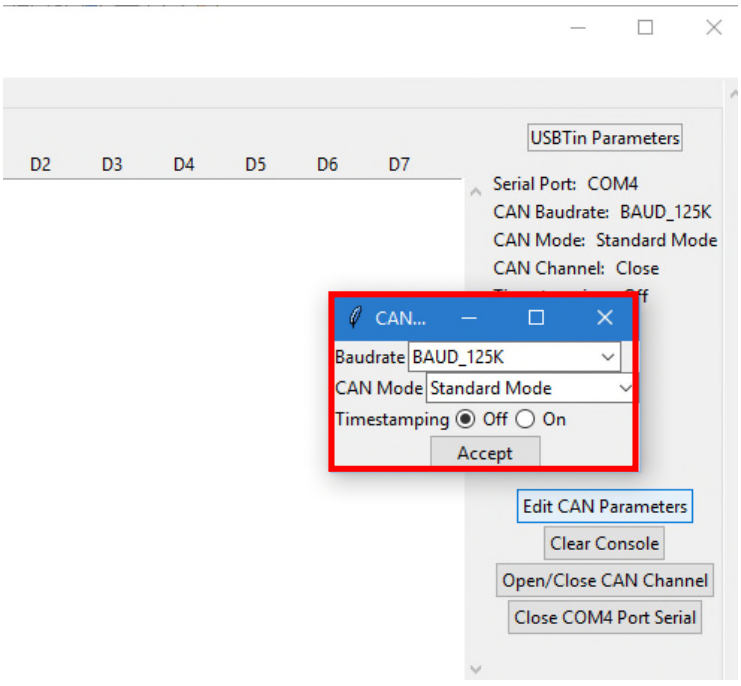


FIGURA 10 - Finestra Subclasse EditParameters. Font pròpia.

5.4.6.2. Subclasse Stats

El continguts de la instància Stats són simplement etiquetes que donen informació sobre l'USBtin i s'actualitzen a mesura que es produeixen esdeveniments gràcies al mètode update definit per aquesta subclasse.

Les informacions que apareixen són el port al que està connectat (també mostrat a la etiqueta del contenidor principal de USBTinFrame), la velocitat del bus CAN ("CAN Baudrate"), el mode del node respecte el canal ("CAN Mode"), l'estat obert/tancat del canal ("CAN Channel") i l'activació/desactivació de l'empremta de temps ("Timestamping").

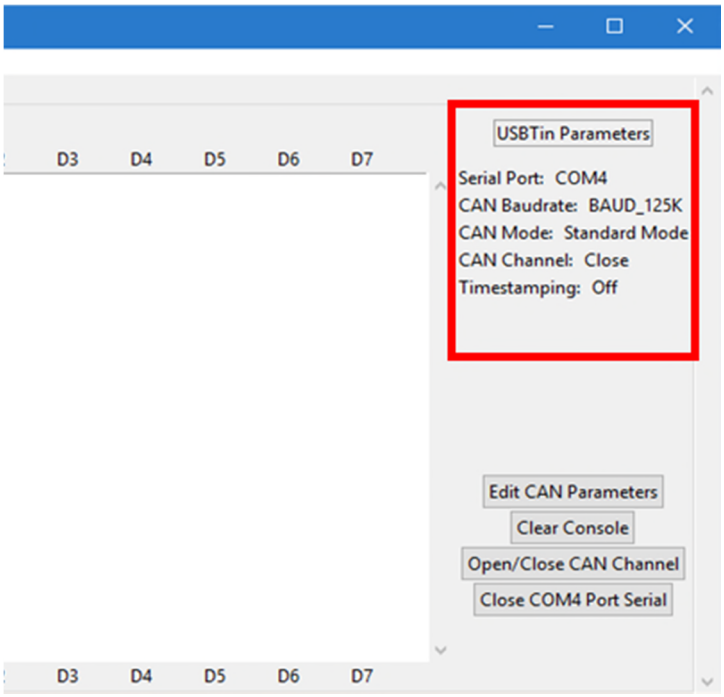


FIGURA 11 - Contenidor Subclasse Stats. Font pròpia.

5.4.6.3. Subclasse ReadMessageFrame

La subclasse ReadMessageFrame és una versió modificada de la classe textwidget d'en Manuel Moreno.

S'ha modificat la línia del títol així com s'han afegit línies per indicar el format dels missatges rebuts (en cas que el que es mostri per pantalla sigui un missatge provinent del bus CAN). A més s'ha vinculat la pantalla amb les diferents funcions desitjades: mostrar els missatges rebuts i transmesos o mostrar les respostes de l'USBtin (demanades als botons del contenidor de la subclasse MenuCANMessage). S'han dissenyat mètodes d'altres classes i subclasses que, quan són cridats, escriuen mitjançant el mètode ReadMessageFrame,writenewline.

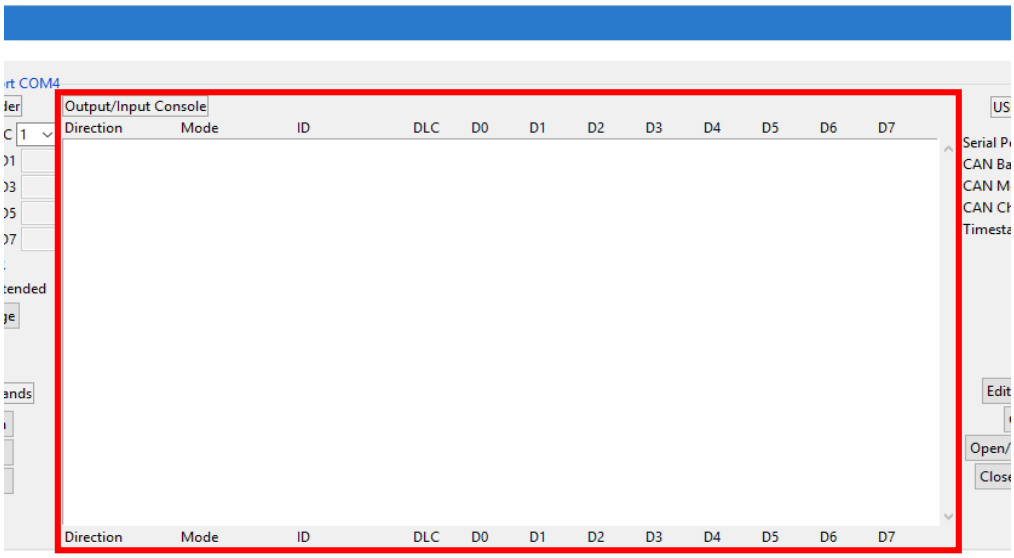


FIGURA 12 - Contenidor Subclasse ReadMessageFrame. Font pròpia.

5.4.6.4. Subclasse MenuCANMessage

La subclasse MenuCANMessage és la que porta la interacció de l'usuari amb la comunicació PC-USBtin-CAN bus.

En un primer terme es troba un menú multi-entrada que s'ha dissenyat a partir de la classe menumultientry de Manuel Moreno, però s'ha modificat per tal que només estiguin actius els bytes (D0...D7) que corresponguin al DLC. També es troben dos parells de botons excloents (Radiobuttons) que permeten seleccionar si el missatge és bàsic o RTR i si és de identificador estàndard o estès. Les dades s'han d'introduir seguint un patró:

- Identificador (*Id*): Sempre s'ha d'introduir amb els dos primer caràcters "0x"; si no, es perdrà informació. A continuació es completa amb l'identificador en format hexadecimal – 0 a 7FF (identificador estàndard) o 0 a 1FFFFFFF (identificador estès). Poden quedar caràcters per definir ja que es completarà automàticament amb zeros (p. ex. si es defineix l'identificador estàndard 0xA, serà igual a definir 0x00A).
- Bytes de dades (D0...D7): Sempre que els apartats de les entrades estiguin actius (segons el DLC - la longitud de bytes de dades seleccionada) es podran introduir aquests bytes en format hexadecimal – de 0 a FF - sempre mantenint, igual que en el cas anterior, els caràcters "0x". També de la mateixa manera que amb l'identificador es completa automàticament amb zeros la informació restant (p. ex. 0xA=0x0A).

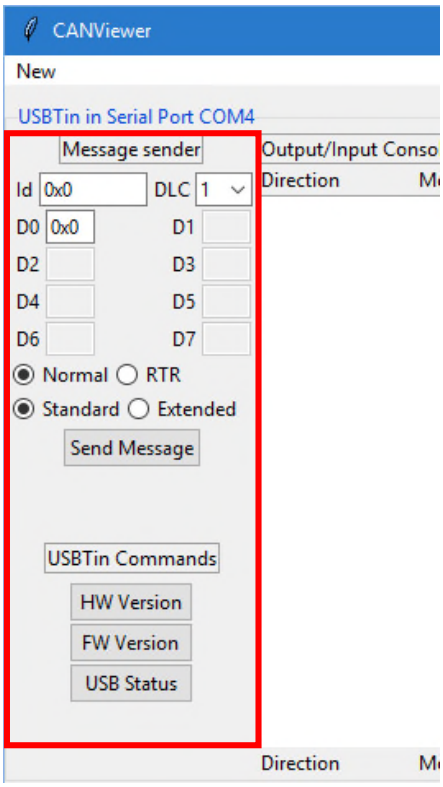


FIGURA 13 - Contenidor Subclasse MenuCANMessage. Font pròpia.

5.4.7. Classe TopLevelMsg

Aquesta classe s'inicialitza en determinades condicions, tal i com s'exposa a continuació. Consisteix en la creació d'una nova finestra amb un missatge d'avís diferent (la inicialització de la classe permet definir diferents missatges). Aquesta finestra compta amb el botó OK, que destrueix la finestra d'avís quan es prem. També es pot definir si al prémer aquest botó també destrueix el contenidor al qual es troba "adjunt". Per exemple, en el cas que no hi hagi ports disponibles i s'intenti inicialitzar la instància USBTinFrame, salta un avís en pitjar el botó "Accept" del contenidor de la instància de NewCOMTop. Quan es clica el botó "OK" de l'avís, a més de destruir la seva pròpia finestra també elimina la de la instància NewCOMTop.

Hi ha tres avisos en diferents situacions:

- *Avís de canal de CAN tancat*: Es produeix quan s'intenta enviar un missatge (pitjant el botó "Send Message" del MenuCANMessage estant la comunicació amb el bus CAN del node tancada. No destrueix cap contenidor més enllà de la finestra del propi avís.

- *Avís de canal de CAN obert:* Es produeix quan s'intenta obtenir una resposta de l'USBtin (pitjant els botons "HW Version", "FW Version" o "USB Status" del MenuCANMessage estant la comunicació amb el bus CAN del node oberta (cal recordar que dins la pròpia biblioteca USBtin no està permès fer aquest tipus de peticions si el canal CAN no està tancat). No destrueix cap contenidor més enllà de la finestra del propi avís.
- *Avís d'intent incorrecte d'obertura de ports*: Es produeix quan s'intenta inicialitzar la instància USBTinFrame i els seus contenidors sense haver cap port disponible per fer-ho. No destrueix cap contenidor més enllà de la finestra del propi avís.

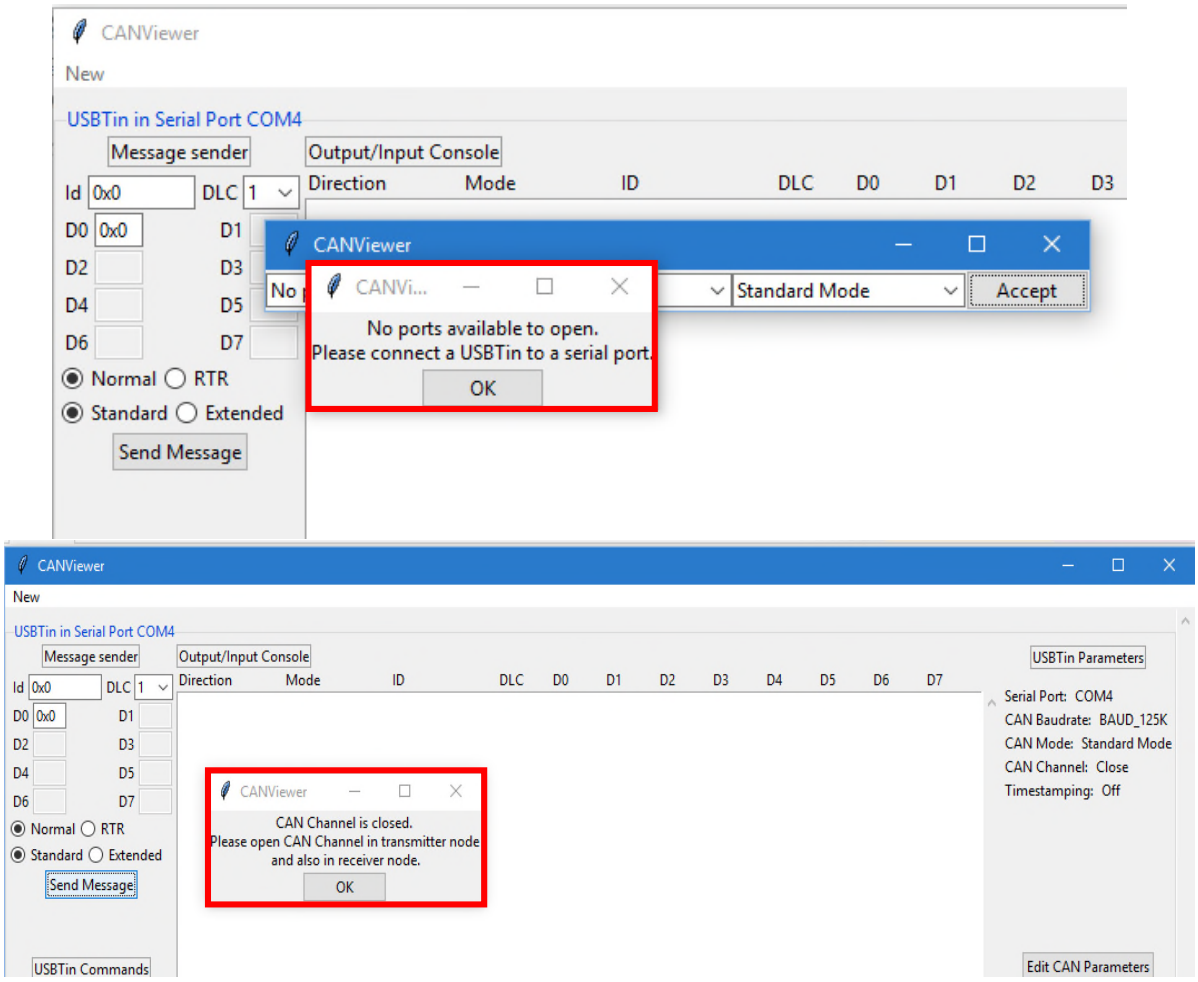


FIGURA 14 - Finestres Classe TopLevelMsg. Font pròpia.

6. Documentació amb Sphinx

Un dels objectius del projecte ha estat el de proveir l'usuari d'una documentació clara i concisa. S'ha documentat completament la biblioteca usbTinLib i els seus mòduls complementaris. . Per fer accessible a més usuaris l'ús de la biblioteca i la seva documentació, l'idioma utilitzat ha estat l'anglès. Per a la creació de tota la documentació s'han seguit dos passos: documentar internament tots els arxius i generar la documentació amb Sphinx.

6.1. Documentació als arxius

El primer pas ha estat descriure dins el propi codi la classe USBTin i tots els seus mètodes de manera que una part de la documentació la tinguéssim també dins els mòduls. Per fer-ho s'han utilitzat *docstrings* dins cada mètode a descriure. S'ha seguit un esquema comú entre tots ells i amb les documentacions de funcions habituals en altres projectes. Primer, es fa una petita descripció del comportament del mètode. Seguidament s'enuncien els paràmetres de la funció i el seu tipus (string, enter, booleà, etc.). Per últim s'indica què retorna la funció o s'indica en quins casos es pot donar un error/excepció. És important indicar que "self" no es comptabilitza com a paràmetre del mètode i hauríem d'indicar que al mètode no se li ha de passar cap paràmetre (None). D'igual manera si un mètode executa una sèrie d'instruccions sense retornar res s'indicarà també a la documentació. A continuació un breu exemple de la documentació en el codi del mètode (al punt 6.3 s'han eliminat tots els *docstrings* per sintetitzar la memòria):

```
def canSetBitrate(self, baudrate):
    """
    Sets baudrate for the CAN channel. It must be set before CAN
    channel is opened.
    By default, canSetBitrate is called inside openSerial function
    (with self.baudrate as parameter).

    Parameters:
        - baudrate: Defined by the keys in dictionaries BAUD or
        canBITRATE

    Returns:
        None

    Raises:
        ValueError - if the baudrate value is not Standarized    nor
        contained in baudrate dictionaries from constantsUSB
    """
```

6.2. Creació de la documentació amb Sphinx

Un cop realitzada la documentació de cada mètode dins de cada arxiu es procedeix a utilitzar l'eina Sphinx per a la creació d'una documentació més accessible i visual. Aquesta eina utilitza un llenguatge de marques (reStructuredText) en la creació dels seus arxius que permeten la seva exportació a diferents formats tan diferents com l'HTML o el PDF, entre d'altres. En el cas d'aquest projecte s'ha escollit el primer.

El primer pas per la creació de la documentació és bastant automàtic, ja que està disponible una comanda des del terminal (sphinx-quickstart) que ajuda a reduir el temps en la creació dels arxius bàsics del projecte (la pàgina mestre index.rst, l'arxiu compilador make, l'arxiu de configuració, etc.) i només cal indicar certes opcions del projecte: nom, autor, versió, extensions utilitzades (important marcar l'extensió autodoc), etc.

Un cop acabada la configuració prèvia es creen els arxius .rst (que són arxius de text indicant que s'està utilitzant el llenguatge de text reestructurat). Per cada pàgina que obtindrem en el resultat final s'ha de crear un arxiu individual i es relacionen entre si incloent un arbre de continguts (toctree) en aquells documents que tenen branques d'arxius. Mirant la figura 15, els arxius que han d'incloure un toctree són index, Class USBTin i Other.

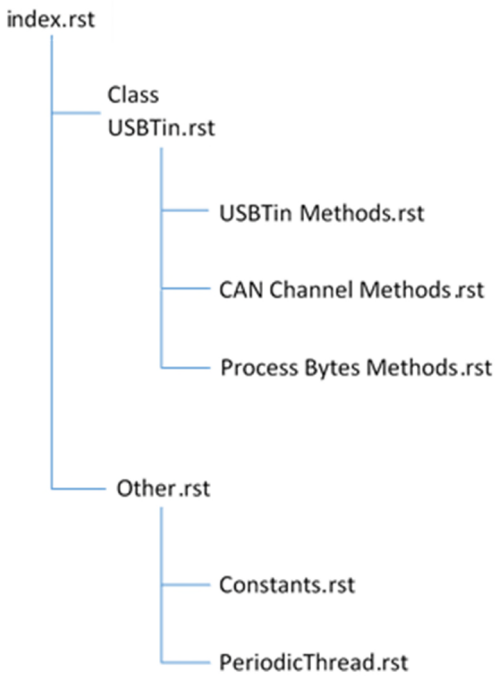


FIGURA 15 – Arbre de continguts de la documentació. Font pròpia.

Dins de cada arxiu, a més del toctree, s'han inclòs els continguts procedents. Gràcies a l'extensió autodoc abans esmentada, s'ha pogut importar tots els *docstrings* de cada mètode de manera que s'ha agilitzat el procés. A més, s'han afegit avisos i exemples de determinats

mètodes. A la figura 15 es mostra la organització interna dels arxius. No obstant això, el que apareix a la taula de continguts de la web són els títols de capçalera a l'interior de cada arxiu de manera que la representació que es veurà a la web serà:

1. *Classe USBTin*
    - *1.1. Mètodes referits a l'USBtin*
    - *1.2. Mètodes referits al canal CAN*
    - *1.3. Mètodes referits al processament de bytes*
2. *Mòduls complementaris*
    - *2.1. Mòdul ConstantsUSB*
    - *2.2. Mòdul PeriodicThread*

Com es pot veure, es un esquema semblant al que s'ha seguit en la construcció d'aquesta memòria.

L'últim pas un cop creats els arxius és compilar-los de manera que es generin els arxius html. Això es fa des del terminal del PC mitjançant la comanda "make html", havent seleccionat la temàtica prèviament a l'arxiu de configuració. S'ha escollit per aquest projecte una temàtica molt similar a la que fa servir la biblioteca PySerial en la seva documentació.

Tot i que s'ha necessitat certa dedicació (sobretot per assolir els detalls que té el llenguatge reStructuredText), la automatització en la creació de pàgines web a partir dels arxius de text redueixen considerablement el temps despès. Sphinx és una eina molt útil per donar forma a la documentació. No obstant això, es va dubtar de bon principi el seu desenvolupament en Windows donat que és menys amigable que altres sistemes operatius com Ubuntu, amb el treball amb comandes a través del terminal.

7. Test i validació

El test i la validació del treball s'ha produït durant tot el desenvolupament del projecte, perquè molts mòduls s'han dissenyat seguint un mètode de prova i error. El test definitiu ha de servir per confirmar el funcionament en un molt ampli ventall de casos per donar validesa definitiva als mòduls. A continuació s'indicarà en diferents apartats els tests que s'han anat realitzant a mesura que s'avançava amb el projecte.

A cada test s'ha provat tant la biblioteca usbTinLib com l'App GUI CANviewer (quan s'ha acabat) en diferents entorns. A la primera s'ha seguit l'esquema del doctest (disponible a l'annex) comprovant que apareixen totes les excepcions mentre que a la segona s'han provat totes les funcionalitats dels diferents widgets i que el seu comportament sigui l'esperat.

7.1. USBtins enfrontats

Per enfrontar els USBtins s'entén connectar-los a diferents ports USB de manera que simulin dos ports sèrie diferents però unint-los en un sol bus CAN de manera que els diferents dispositius siguin nodes d'un sol bus. Amb aquesta configuració s'han realitzat diferents proves que s'exposaran a continuació.

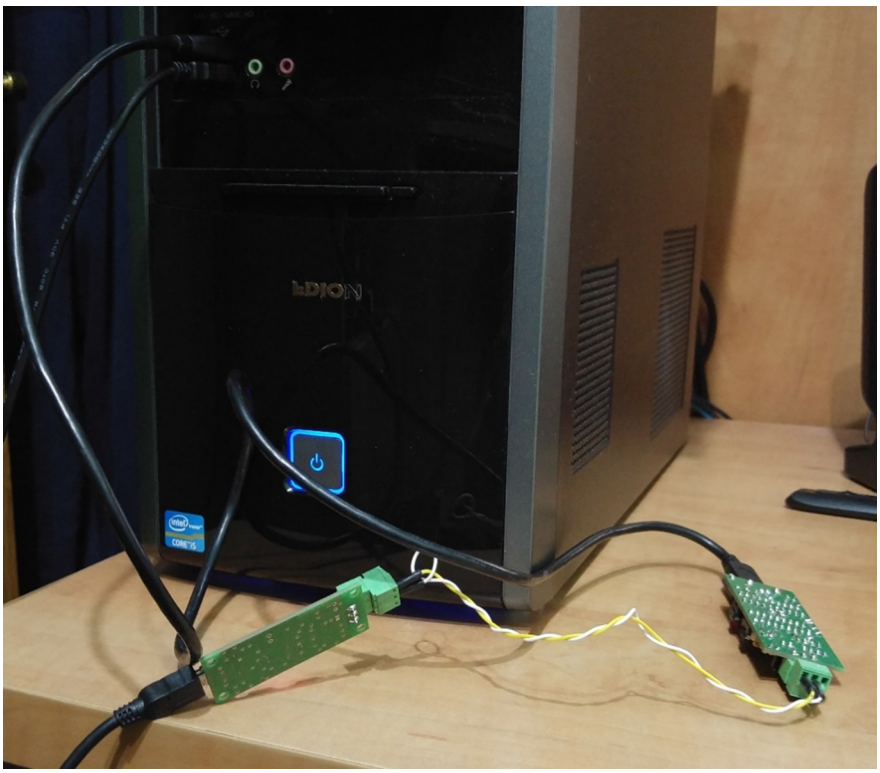


FIGURA 16 – Configuració amb els USBtin enfrontats. Font pròpia.

7.1.1. Biblioteques pròpies

Ha consistit en utilitzar únicament els mòduls relacionats amb el projecte de manera que si es volia comprovar la biblioteca usbTinLib es creaven dues instàncies diferents on variàvem paràmetres com el port a obrir o el mode del canal CAN. Si es volia provar amb l'aplicació CANviewer aquesta ja dóna facilitats (es una de les raons de la seva creació) per treballar amb dos o més ports.

Aquest procediment ha estat el més utilitzat durant el projecte perquè facilitava la detecció i correcció d'errors gràcies a la disponibilitat de dos dispositius.

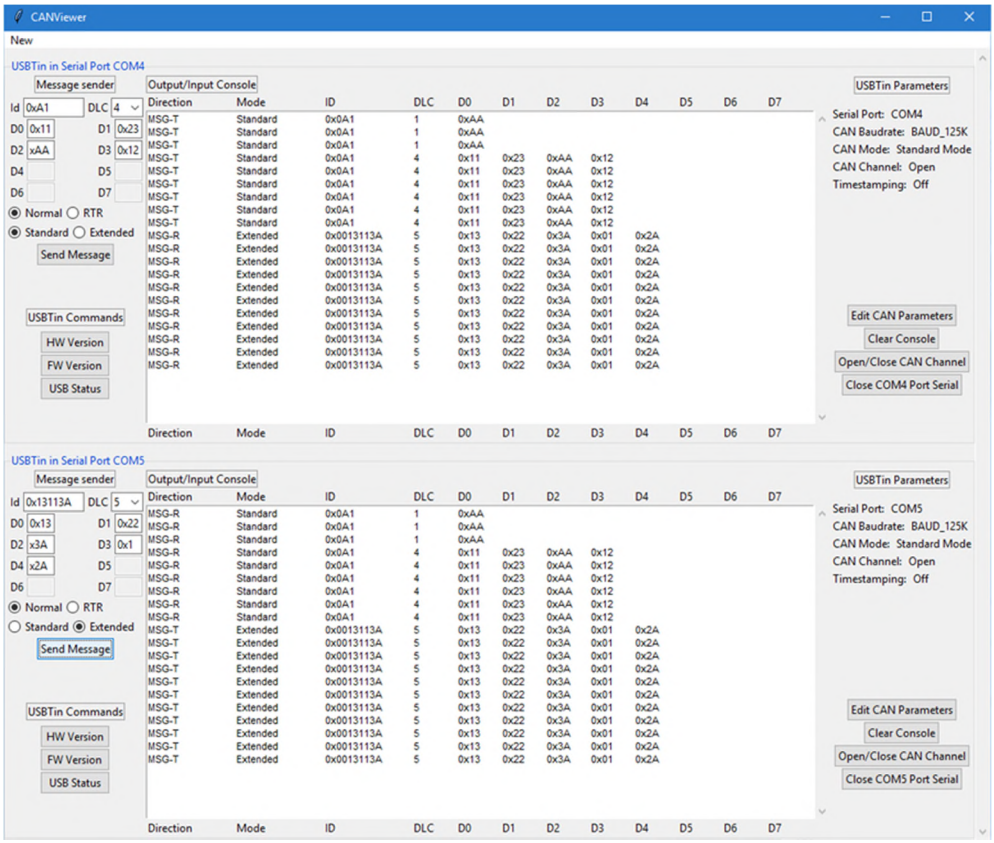


FIGURA 17 – CANviewer GUI amb dos USBtin enfrontats. Font pròpia.

7.1.2. Biblioteques diferents

El test amb altres biblioteques s'ha realitzat sempre mitjançant les GUI respectives i ha estat una de les etapes finals i de depuració. Mantenint la configuració dels USBtin abans esmentada, s'ha obert un dels dispositius amb l'aplicació CANviewer del projecte i l'altre amb la GUI USBtinViewer_v1.2 feta pel creador de l'USBtin, Thomas Fischl.

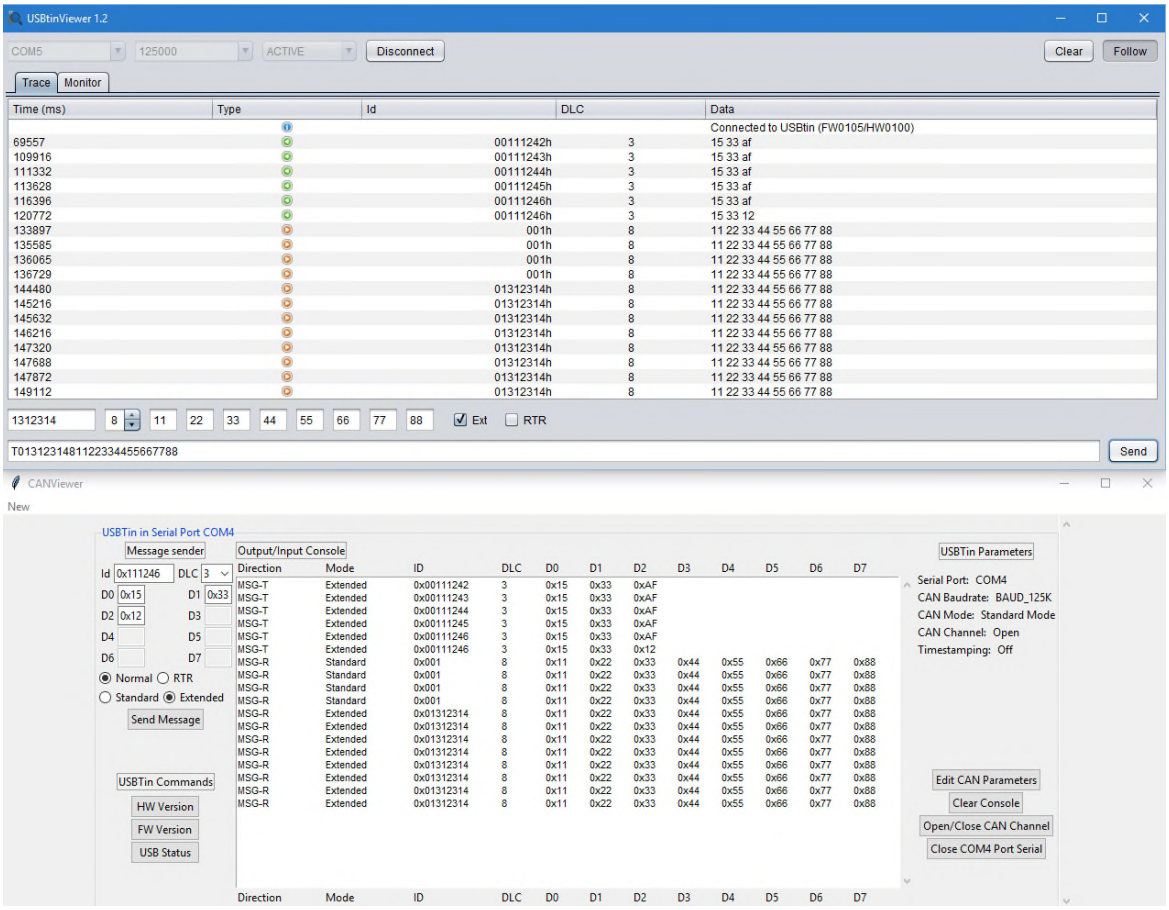


FIGURA 18 – CANviewer i USBtinViewer treballant sobre dos USBtin diferents. Font pròpia.

7.1.3. Generador automàtic de missatges

Per acabar de validar la biblioteca i demostrar la utilitat s'ha decidit crear un generador automàtic de missatges CAN. Coneixent a quin port sèrie està connectat l'USBtin que farem servir com a generador, es dissenya el codi per tal que s'enviïn cada centèsima de segon un total de 100 missatges. Aquí es demostra la gran utilitat de la biblioteca en poder crear un generador de missatges amb unes poques línies.

```
import usbTinLib as usb
import time

usb1=usb.USBTin('COM5','BAUD_125K','Normal')
usb1.canOpenChannel()

for i in range(100):
    usb1.canWrite('131',4,['12','65','77','AA'],0x0002)
    time.sleep(0.01)

usb1.canClose()
```

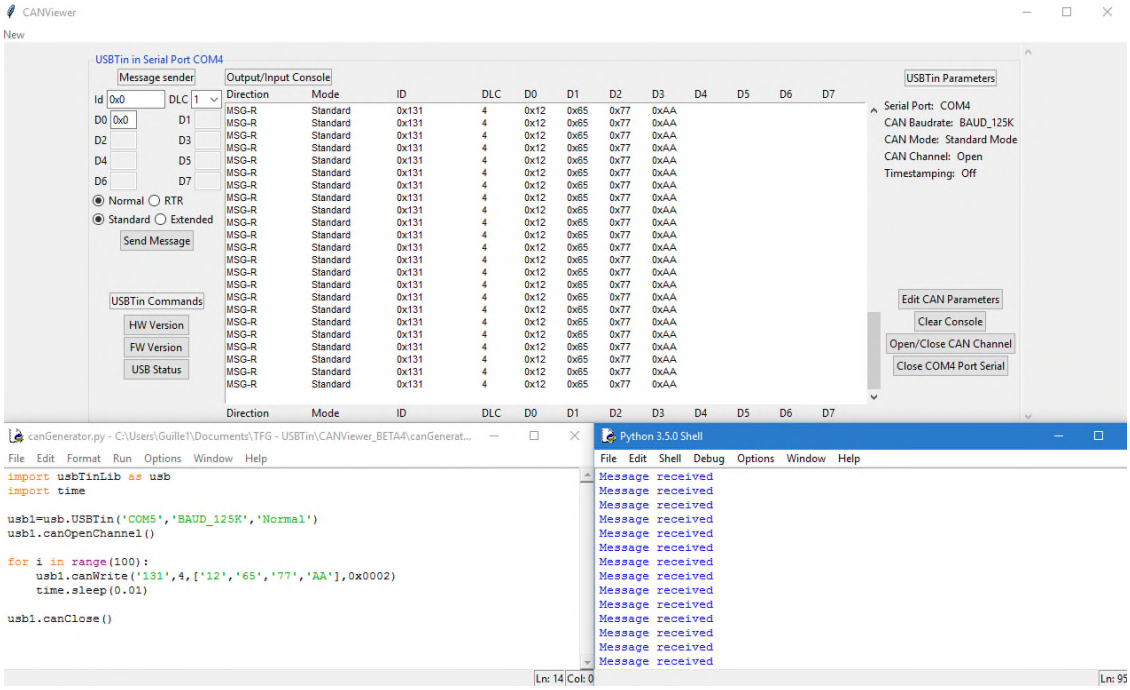


FIGURA 19 – Generador de missatges al 'COM5' visualitzat amb CANviewer. Font pròpia.

Aquest test ha estat molt útil per optimitzar els mètodes de la biblioteca i el processament de les dades ja que en els primers tests s'observava que els missatges es processaven molt lentament i pel node observador es llegien missatges que havien estat enviats segons abans.

7.1.4. Diferents S.O./versions de Python

Tota la biblioteca i l'aplicació ha estat dissenyada en Python3. Per tant, és incompatible amb Python2 en gran part de la sintaxis del codi i no és possible executar el programa amb l'intèrpret d'una versió més antiga.

La compatibilitat amb els sistemes operatius, doncs, ve determinada en gran mesura per la pròpia compatibilitat de Python. Per sort, l'intèrpret és compatible amb els tres S.O. més utilitzats: Windows, Linux i MacOS.

A la conclusió d'aquest document només s'ha pogut comprovar la correcte execució en Windows 8 i 10. En Linux s'ha comprovat que l'aplicació CANviewer s'inicia correctament però un petit problema en la detecció del port sèrie creat en connectar el dispositiu USBtin no ha permès realitzar el test de la funcionalitat completa.

7.2. USBtin enfrontat a Kvaser Leaf Light

L'aparell de Kvaser també és un adaptador USB-CAN que té la mateixa funcionalitat que l'USBtin, però més desenvolupat al ser un hardware desenvolupat per una empresa amb més recursos. El test amb el Kvaser ha estat el més definitiu i ha posat a prova tota la biblioteca.

La configuració del test va ésser una mica complexa, ja que es van connectar un total de quatre dispositius - tres USBtin i un Kvaser Leaf Light – de la següent manera. Compartint un bus CAN es van enfrontar dos USBtin i compartint un altre bus CAN diferent es van enfrontar un USBtin i el Kvaser. Així es podia comprovar el funcionament de tota la biblioteca amb diferents busos actius.

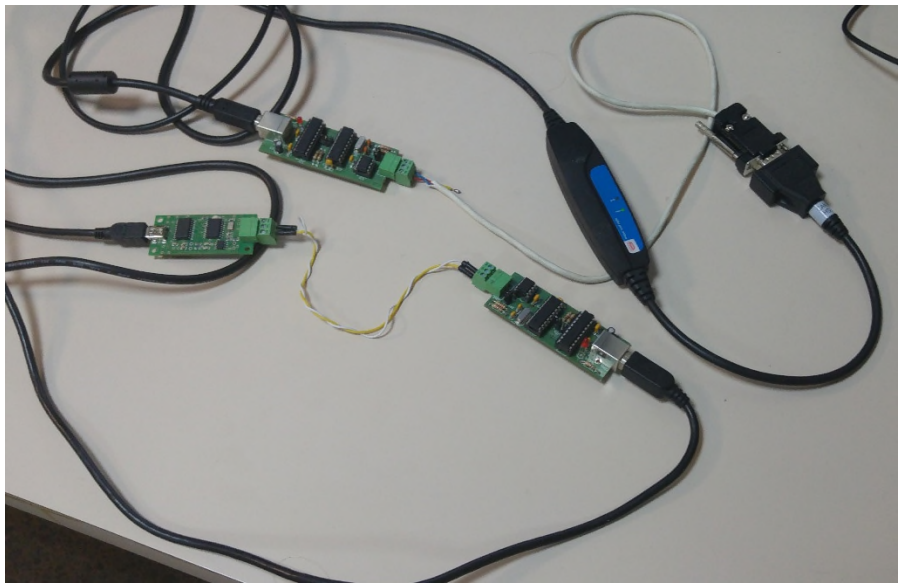


FIGURA 20 – Configuració del test final. Font pròpia.

A partir de la configuració, mitjançant el software Kvaser CAN King s'han generat missatges al bus CAN i s'ha comprovat la seva recepció al node observador mitjançant CANviewer ('COM14' a la figura 20) així com s'han enviat missatges des del CANviewer per comprovar la seva correcte transmissió al bus

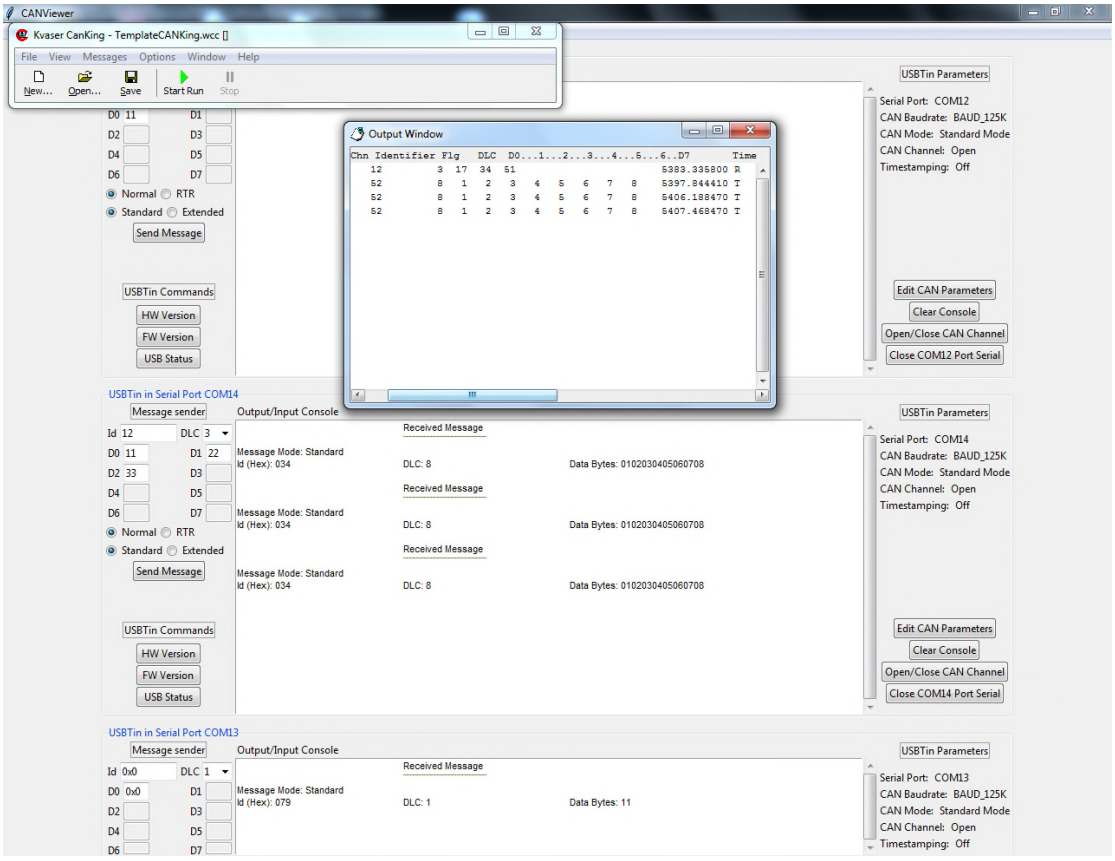


FIGURA 21 – Enviament i recepció de missatges CANviewer-Kvaser CAN King. Font pròpia.

Aquest test es va dur a terme amb una versió no finalitzada completament de la biblioteca usbTinLib i la GUI (per aquesta raó s'observen diferències amb altres figures), però els canvis realitzats no afecten a la seva funcionalitat.

8. Pressupost

Es consideren les despeses derivades tant dels recursos humans (degut al disseny i desenvolupament de l'aplicació) com dels components i dispositius necessaris per al desenvolupament de l'aplicació. No es consideraran, doncs, els dispositius que s'han utilitzat únicament al test de l'aplicació.

Activitat	Hores dedicades	Preu/hora	Cost total
Estudi preliminar	40 h	20€/h[3]	800 €
Desenvolupament de la biblioteca usbTinLib i mòduls complementaris	80 h	50€/h	4000 €
Desenvolupament de la GUI CANviewer	70 h	50€/h	3500 €
Desenvolupament de la documentació amb Sphinx	40 h	50€/h	2000 €
Test dels mòduls	20 h	50€/h	1000 €
Redacció de la memòria	50 h	20€/h	1000 €
Total	300 h	-	12300 €

TAULA 2 – Costos del projecte derivats dels recursos humans

Per la seva banda el cost degut als components utilitzats és:

Component	Quantitat	Preu unitari	Cost total
USBtin	2	34,90 €/u	69,80€
Adaptador USB mascle mini-A/tipus A	1	10€/u	10€

[3] Els preus són una aproximació i queden fixats pels honoraris, amortització de materials, despesa energètica, benefici de l'empresa adjudicataria del projecte, etc.

Adaptador USB mascle tipus B/tipus A	1	10€/u	10€
Cables per simular el bus	2 x 0,2 m	<1€/m	≈ 0 €
Total	-	-	89,80 ≈ 90 €

TAULA 3 – Costos del projecte derivats dels components

El cost total del projecte ha estat de **12300 €.**

9. Impacte medi ambiental

L'impacte medi ambiental d'un projecte com aquest, en què l'objectiu ha estat el desenvolupament de biblioteques i aplicacions informàtiques, és molt complicat de comptabilitzar. Qualitativament, però, es pot assegurar que l'impacte és molt baix. Això és degut a que l'únic àmbit que té un cert marge de maniobra seria el de la despesa energètica.

Si considerem la despesa energètica derivada del disseny i la creació del projecte es pot concloure que l'impacte ha estat mínim ja que s'ha treballat des d'un ordinador de sobretaula amb un consum mig de 300-350 W. Si considerem els temps indicats en l'apartat precedent, la despesa energètica total derivada del disseny de l'aplicació ha estat 350 W x 300 h = 105 kWh =$3{,}78 \cdot 10^8$ J. Això representa aproximadament l' 1 % del consum mitjà d'un habitatge **[10].** Tot i que és una xifra representativa si la comparem amb un consum a nivell d'usuari, no ho és tant en el context del desenvolupament d'un projecte.

També es podria estudiar la despesa energètica a nivell electrònic dels components de l'USBtin i la seva interacció amb el PC mitjançant la biblioteca i aplicació dissenyada però hi ha massa variables per avaluar-ho a nivell quantitatiu (nombre d'usuaris, nombre de dispositius per usuari, temps mig de funcionament, etc.). Des d'un punt de vista qualitatiu es pot assegurar que el consum dels components microelectrònics es prou baix com per assegurar que l'impacte en aquest sentit és menyspreable.

Conclusions

Els objectius del projecte s'han complert plenament. S'ha dissenyat la biblioteca usbTinLib, que inclou mètodes per a la majoria de funcions de l'USBtin i la interfície gràfica que els utilitza, dotant-los d'una major presència visual i gràfica per a l'usuari. També la documentació del projecte ha estat un punt important per acomplir l'objectiu de realitzar una biblioteca i una aplicació accessible i que possibiliti el seu ús en futurs projectes.

La biblioteca i l'aplicació creades, tot i no ser exigents quant a coneixements profunds de la ciència de computadors, han requerit de l'ús de múltiples mòduls. Això ha ajudat a expandir els coneixements sobre els diferents mòduls en concret i de Python en general. Un exemple a mencionar ha estat el de la creació de l'aplicació. Es va començar sense conèixer res del mòdul tkinter i s'ha acabat pràcticament sense consultar la documentació. Això demostra que l'accessibilitat que s'havia previst és real gràcies a la ràpida corba d'aprenentatge en un llenguatge com Python.

Queda una biblioteca útil que pot ser combinada amb altres mòduls que realitzin funcions complementàries: traducció i descodificació dels missatges CAN, agrupació per identificadors, etc. Un ventall molt ampli de possibilitats que es podran desenvolupar fàcilment gràcies a la feina feta amb la claredat i la documentació del codi.

Agraïments

A en Manuel Moreno, per la seva ajuda durant tot el projecte, ja sigui en forma de mòduls d'exemple o participant en la validació de la biblioteca i l'aplicació.

A en Marcel Garrido i a en Thomas Fischl, per les seves biblioteques que han servit de referència en el disseny de la nova biblioteca usbTinLib.

A Chris Liechti, Kris Dorosz i Thomas Feldmann, pels seus mòduls o biblioteques externes de codi lliure que han estat d'utilitat (amb o sense modificacions) en el desenvolupament del projecte.

Bibliografia

Referències bibliogràfiques

[1] FISCHL, THOMAS. USBTin. A: Thomas Fischl, USBtin project [en línia]. Actualitzat 2015. [Consulta 10 de gener de 2016]. Disponible a <http://www.fischl.de/usbtin/>

[2] LIECHTI, CHRIS. PySerial. A: Readthedocs [en línia]. Actualitzat 2015. [Consulta: 24 de desembre de 2015]. Disponible a <https://pyserial.readthedocs.org/en/latest/>

[3] Tkinter Documentation. A: Tk Commands, version 8.6.4 [en línia]. [Consulta: 24 de desembre de 2015]. Disponible a <http://www.tcl.tk/man/tcl8.6/TkCmd/contents.htm>

[4] DOROSZ, KRIS (usuari cypreess). PeriodicThread module. A: Github [en línia – fitxer Python]. Fitxer actualitzat Abril 2013. [Consulta: 24 de desembre de 2015]. Disponible a <https://gist.github.com/cypreess/5481681>

[5] FISCHL, THOMAS. USBtinLib 1.1.0. A: Github [en línia – múltiples fitxers Java]. Fitxers actualitzats Octubre 2015. [Consulta: 5 de gener de 2016]. Disponible a <https://github.com/EmbedME/USBtinLib>

[6] FELDMANN, THOMAS. Listing Serial Ports. A: Stackoverflow [en línia]. Gener 2013, actualitzat Novembre 2015. [Consulta: 5 de gener de 2016]. Disponible a <http://stackoverflow.com/questions/12090503/listing-available-com-ports-with-python>

[7] VerticalScrolledFrame. A: Unpythonic [en línia – fitxer Python]. Fitxer actualitzat Juliol 2010. [Consulta: 5 de gener de 2016]. Disponible a <http://tkinter.unpythonic.net/wiki/VerticalScrolledFrame>

[8] MORENO, MANUEL. textwidget [Fitxer Python]. Barcelona: EEL-UPC, Març 2013. Actualització Abril 2013. Class textwidget.

[9] MORENO, MANUEL. EEL-UPC. menumultientry [Fitxer Python]. Barcelona: EEL-UPC, Març 2013. Actualització Abril 2013. Class Menumultientry.

[10] OCU. ¿Cuanta energia consume una casa? A: OCU. Organización de Consumidores y usuarios [en línia]. Madrid: OCU [Consulta: 5 de gener de 2016]. Disponible a <http://www.ocu.org/vivienda-y-energia/gas-luz/noticias/cuanta-energia-consume-una-casa-571584>

Bibliografia complementària

[11] GARRIDO, MARCEL. Interfície gràfica d'usuari usant una tauleta d'Android i un adaptador CAN / USB. Barcelona, 2015. [Treball de Final de Grau]

[12] KLEIN, BERND. Threads in Python. A: Python Course [en línia]. Actualitzat 2015. [Consulta: 31 de desembre de 2015]. Disponible a <http://www.python-course.eu/threads.php>

[13] Kvaser CANLIB. A: Kvaser [en línia]. Actualitzat Desembre 2015. [Consulta: 5 de gener de 2016]. Disponible a <http://www.kvaser.com/canlib-webhelp/group___c_a_n.html>